An error-tolerant graph matching model for hierarchical graphs with topological node embedding in large scale visual retrieval

Pau Riba

Abstract

Graph-based representations are effective tools to capture structural information from graphical elements. However, retrieving a query graph from a large dataset of graphs implies a high computational complexity. Moreover, these representations are very sensitive to noise or small changes. The most important property for a large-scale retrieval is the search time complexity to be sub-linear in the number of database examples. In this work, a novel hierarchical graph representation is designed. Using graph clustering techniques adapted from graph-based social media analysis, we propose to generate a hierarchy able to deal with different levels of abstraction while keeping information about the topology. An embedding function is designed to summarize the local node context in terms of topological features. At the abstract levels of the hierarchy, these features encode relations between clusters. The proposed embedding is used to enrich the information codified in all the nodes. For indexation purposes in large scale scenarios, a graph hashing architecture is proposed by reducing the topological node features to a binary embedding. For the proposed representations, a coarse-to-fine matching technique is defined for the hierarchical one and a graph indexation formalism easily computed by means of bitwise logical operators is presented for the local context embedding enrichment. The proposed approaches are validated in different real scenarios such as handwritten word spotting in images of historical documents and classification of color images. Moreover, a synthetic dataset has been created to validate the proposed approach in a controlled scenario.

Index Terms

Structural pattern recognition, Graph-based representations, Graph hierarchy, Graph indexation, Graph embedding

I. Introduction

CONTENT-BASED image retrieval (CBIR) systems [1] have become more complex in the recent years with the increase of data spread in the cloud (e.g. image repositories and videos in social networks). The practical success of machine learning methods applied to simple image representations for retrieval or recognition has faded away other schemes representationally richer but practically infeasible. However, to tackle with complex recognition problems, methods not exclusively based on appearance but enriched with more abstract visual information, such as the visual structure of objects, are required. *Structural pattern recognition* uses symbolic representational models such as strings, trees, graphs, hypergraphs etc. to describe visual objects. These symbolic representations encode spatial, temporal, hierarchical or conceptual relations between primitives. Although the first attempts of part-based descriptors suggesting graph representations were presented long ago [2], it has been in the last decade when structural models have gained importance in computer vision. Structural representations are implicitly or explicitly drivers of more powerful approaches for visual recognition and retrieval than statistical approaches. Graph-based representations are able to deal with many-to-many relationships among visual features and their parts.

Graph-based representations can play an important role in CBIR. Using these representations, not only statistical information is codified but also the relations between the compounding parts. Graph representations in computer vision have two main requirements. Firstly, the underlying structures have to be extracted from the dataset images. Afterwards, comparison tools able to deal with deformations must be provided, for instance, error-tolerant graph matching approaches. In this scenario, the number of graphs in the database may be extremely large, and also the graphs may be large. Although many suboptimal methods for graph matching exist, it is unfeasible to compare a query graph with thousand or million graphs of the database in a sequential way.

Many works have proposed error-tolerant graph matching techniques. However, very little research has been done to face graph matching in large scale scenarios. To deal with large-scale problems, methodologies able to prune nonpromising graphs must be developed. These methodologies must be efficient in terms of space and time. One of the contributions of this work is a graph representation able to discard those non-promising structures.

In this thesis, graph-based representations are enriched to deal with error-tolerant graph matching in large scale scenarios. Firstly, a graph hierarchy representation by means of a contraction function is used to perform a matching in abstract representations. For each level in the hierarchy, the graph size is reduced allowing a fast matching able to prune the comparisons

Author: Pau Riba, priba@cvc.uab.cat
Advisor 1: Josep Lladós, CVC, UAB
Advisor 2: Alicia Fornés, CVC, UAB
Thesis dissertation submitted: September 2016

using more detailed graph levels. Moreover, given a labeled graph, each node is enriched with topological node features with information about their local context. Binarizing these local context embedding, a fast indexation scheme is proposed, which is able to avoid many unnecessary comparisons. The most important property for a large-scale indexation scheme is the search time complexity to be sub-linear in the number of database examples.

The rest of this dissertation is organized as follows. Section II overviews the relevant methods in the literature. Section III formalizes two graph representation, firstly, a hierarchical graph is presented in order to encode information about different levels of abstraction. Afterwards, a node embedding to codify their local context is developed. Section IV proposes the matching techniques for the proposed representations, a coarse-to-fine matching for the hierarchical representations and an indexation scheme for the local context embeddings. Section V is devoted to evaluating all the proposed techniques. Finally, Section VI draws the conclusions and introduces the future work.

II. State of the art

This section overviews the key references of graph techniques that are relevant to the present work. Firstly, the recent advances of error-tolerant graph matching are presented. In computer vision applications, a distortion model must be incorporated due to the inherent variations of visual objects. Then new approaches based on graph embeddings and graph kernels are studied. These kinds of techniques have emerged rapidly as efficient techniques to incorporate structural properties in machine learning classifiers [3]. Afterwards, indexation and clustering techniques are explained. Finally, different hierarchical representations are reviewed. Let us start defining *attributed graph*.

Definition 2.1 (Attributed Graph): An *attributed graph* G is defined as a 4-tuple $G = (V, E, \mathrm{L_V}, \mathrm{L_E})$ where V is the set of nodes; $E \subseteq V \times V$ is the set of edges; $\mathrm{L_V}$ and $\mathrm{L_E}$ are two labeling functions defined as $\mathrm{L_V} : V \to \Sigma_V \times A_V^k$ and $\mathrm{L_E} : E \to \Sigma_E \times A_E^l$, where Σ_V and Σ_E are two sets of symbolic labels for vertices and edges, A_V and A_E are two sets of attributes for vertices and edges, respectively, and $k, l \in \mathbb{N}$. We will denote the number of vertices in a graph by $|V|$ and the number of edges by $|E|$.

A. *Error-tolerant graph matching*

Graph matching is one of the most important challenges of graph processing. Generally speaking, the problem consists in finding the best correspondence between the sets of vertices of two graphs preserving the underlying structures and the corresponding labels and attributes. Several algorithms have been proposed in the literature [4].

The intrinsic variability of patterns, noise and errors produced from the graph extraction process, makes mandatory to encode tolerance to errors into graph matching frameworks. This approach, called error-tolerant graph matching, measures the similarity between two given graphs. Figure 1 shows the taxonomy of some important graph matching approaches divided into exact or error-tolerant ones.

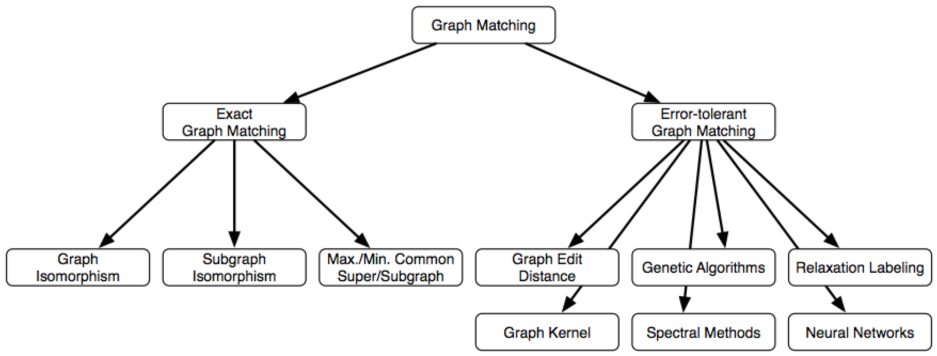


Fig. 1. A taxonomy of graph matching approaches. Figure extracted from [3]

Graph edit distance [5], [6], [7] is the process of evaluating the similarity of two different graphs. The computation of the distance between two graphs is inspired in the string edit distance. Thus, the main idea is to compute the minimum cost edition or transformation from the source graph to the target one in terms of a sequence of edit operations. The typical edit operations are node and edge insertion, deletion and substitution. Each edit operation has an associated cost so every step will add a cost to the final edit distance. The graph edit sequence transforming one graph into the other one is not unique, because of that, the minimum cost path has to be computed. Although the method finds an optimal edit path between two graphs, the computational complexity of the edit distance algorithm is exponential in the number of nodes of the involved graphs. A suboptimal approximation to graph edit distance called bipartite graph matching was proposed by Riesen *et al.* [8]. The algorithm is based on the assignment problem solution using a cost matrix which codifies the edit operations costs. Figure 2 shows a possible edit path between two graphs.

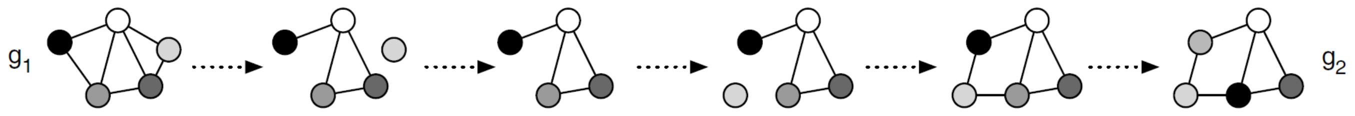


Fig. 2. A possible edit path between graph g_1 and g_2 (node labels are represented by different shades of grey). Figure extracted from [8].

More recently, an efficient approach based on graph factorization applied to deformable object recognition and alignment have been proposed by Zhou and De la Torre [9]. They have formulated the graph matching problem as a quadratic assignment one. They propose to factorize the pairwise affinity matrix into smaller matrices that encode the local structure of each graph and the pairwise affinity between edges. They claim that the proposed factorization avoids the computation of the pairwise affinity matrix. This factorization can be applied jointly with other graph matching methods. Also, it allows the use of a path-following optimization algorithm. Finally, they demonstrate that the given factorization can incorporate a geometric transformation to the graph matching problem.

B. Graph kernels and embedding

Pattern recognition techniques have been usually classified into statistical and structural approaches. Table I shows the main differences between both families of approaches. A '+' denotes an advantageous position in the corresponding feature. An emerging trend is the use of graph kernels and embeddings unifying both methodologies [10]. These two techniques allow the use of the computational tools from statistical pattern recognition with structural data.

TABLE I
STRUCTURAL VS STATISTICAL PATTERN RECOGNITION COMPARISON. THE POSITIVE ASPECTS ARE MARKED USING '+'.

	Pattern Recognition	
	Structural	**Statistical**
Data structure	Symbolic data structure	Numeric feature vector
Representational strength	+	-
Flexible dimensionality	+	-
Robustness to noise	-	+
Efficient computational tools	-	+

Graph embedding has led the research in graph-based pattern recognition in the last years. Informally speaking, graph embedding consists of defining a transformation between the graph space to an n-dimensional numerical space, so that the similarity between two instances is preserved. Consequently, graph distance computation can be solved by a classifier from the machine learning area more efficient computationally.

Definition 2.2 (Graph embedding): A *graph embedding* is a mapping from the set of graphs $\mathcal{G}$ to a vectorial space.

$$\begin{aligned}\phi\colon \mathcal{G} &\to \mathcal{R}^n \\ g &\mapsto \phi(g) = (f_1, f_2, \ldots, f_n)\end{aligned}$$

This methodology overcomes the lack of efficient algorithmic tools that can be applied to graphs. Riesen *et al.* [11] propose an embedding function by means of prototype selection and graph edit distance computation. This embedding consists in computing the edit distance between a query graph and the set of n prototype graphs. Afterwards, an n-dimensional vector is constructed from these distances.

Kernel machines can be adapted to structural pattern recognition providing a powerful tool to compare two graphs. A graph kernel function computes an inner product between graphs giving a measure of similarity between them. It is also called implicit embedding. Several kernel methods have been proposed in the literature. Borgwardt *et al.* [12] propose an extension to the random walk kernel in order to deal with continuous labels. Borgwardt and Kriegel [13] propose a pair of kernel methods (all-paths and shortest-path kernels) that compute the similarity between two graphs by means of similarity between pairs of paths. Harchaoui and Bach [14] propose a tree-walk kernel that counts common virtual substructures between graphs. Their application scenario is the segmentation of natural images using graphs. The presented method allows to perform efficiently supervised classification of natural images with a support vector machine.

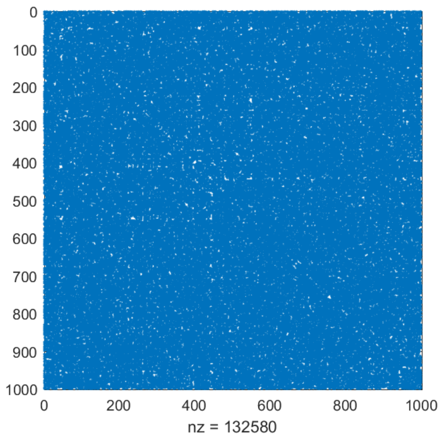
C. Graph indexation

In general, graph indexing is solved by graph factorization techniques where the database of graphs is decomposed in smaller ones that represent a codebook of compounding structures. The indexation is therefore stated in terms of indexing the constituent graphs organized in a lookup table structure. Usually, path-based methods are used to split the graphs into small redundant fragments. *GraphGrep* proposed in [15] that enumerates all the existing paths up to a predefined length. One of the most recent and relevant works in graph indexing was proposed by Yan *et al.* [16]. Frequent graph substructures are obtained by graph sequentialization, according to a *depth first search* (DFS) traversal of the graph edges. Edge sequences are organized in a prefix tree called the *gIndex* tree. The approach is applied to protein graphs.

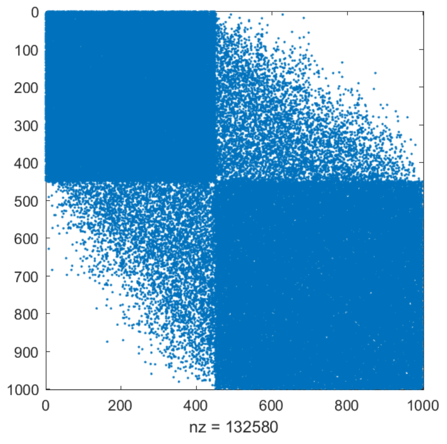
Another approach proposed by Messmer and Bunke [17] consist in organizing the constituent graphs in a decision tree based in decompositions of permutations of the adjacency matrix. At run time, subgraph isomorphisms are detected by means of a decision tree traversal. The complexity for indexing is polynomial in the number of input graph vertices, but the decision tree is of exponential size. A similar approach based on the construction of a graph lattice was proposed in [18]. The performance for large-scale retrieval is achieved by matching many overlapping and redundant subgraphs. Cheng *et al.* in [19] proposed an indexing technique for graph databases. It is based on constructing a nested inverted-index, called *FG-index*, based on the set of frequent subgraphs. The above methods are good for graphs with single labels and without strong distortion degree.

D. Graph clustering

Graph clustering consists in reducing the graph dimension in terms of the similarity function between nodes under a certain topology. An increasing trend to solve graph clustering is to adapt the community search algorithms used in social media analysis. Graph clustering is a way to gather graph vertices into clusters or groups. Therefore, all nodes in a cluster will be related somehow depending on the clustering algorithm, usually, highly connected groups of nodes. Schaeffer [20] presents a survey on graph clustering explaining quality measures for the clusters and how to construct them. Few methods have been reported in the literature providing promising results. The idea of graph clustering is to find an ordering of the adjacency matrix that groups together highly connected sets of nodes. Figure 3 shows two orderings of an adjacency matrix from the same graph. This graph has been generated creating two highly dense communities of sizes 450 and 550 respectively, afterwards, relations between these clusters have been added randomly.



(a)



(b)

Fig. 3. Adjacency matrix of a graph of sizes $|V| = 1000$ and $|E| = 66290$, (a) no clusters can be identified (b) same matrix sorted in the correct way reveals the two communities as two highly dense blocks.

There are two main families of graph clustering algorithms, *global* and *local* ones. Global algorithms require knowledge of the whole graph. This constraint is very costly when working with huge graphs with lots of nodes. Therefore, local methods can be applied only using adjacency lists of a vertex and its neighbors. Despite local techniques are more efficient in terms of time and memory, global approaches perform better in terms of node assignment to the real clusters.

Let us focus on the global algorithms. This kind of techniques assigns a cluster to each node of the graph. Clusters do not need to be rigidly defined, sub-clusters can be present in the graph, for instance, on a real image, people can be segmented (cluster of humans) but also more details can be found (cluster of parts) such as head, arms, legs etc. Plenty of techniques deal with this problem using a hierarchical process that can be represented using a dendrogram. There, the root node corresponds to all the graph, meaning that all the nodes belong to the same cluster. Afterwards, at each level, the nodes are assigned to their corresponding clusters. The leaves mean that each node represents a single node cluster.

Global algorithms can be also divided into *divisive* and *agglomerative* ones. The former start from the entire graph in one cluster and recursively split the graph in a top-down fashion generating the clusters, whereas the agglomerative methods start from an empty set of clusters and iteratively classify the nodes, adding new clusters or joining them.

Girvan-Newman algorithm [21] is a well-known hierarchical method used for community detection in complex systems. It is a global divisive algorithm which removes the appropriate edge at each step until all the edges are deleted. To chose the edge to be deleted, the *Girvan-Newman* algorithm uses the *betweenness centrality* measure of an edge [22]. The *betweenness* *centrality* on an edge $e \in E$ is defined as the number of shortest walks between any pair of nodes that cross e. The idea is that the edges with higher centrality are the candidates to be connecting two clusters. After the edge deletion, each connected component is considered as a cluster in the hierarchy. Note that some iterations will keep the same clustering because no new connected components are created. Finally, the output of this algorithm is a dendrogram. This algorithm consists of 4 steps:

1) Calculate the betweenness centrality for all edges in the network.
2) Remove the edge with highest betweenness and generate a cluster for each connected component.
3) Recalculate betweennesses for all edges affected by the removal.
4) Repeat from step 2 until no edges remain.

E. Hierarchical graph representations

Graph-based representations are a powerful tool to codify the information coming mainly from the structure. However, structural representations are very sensitive to deformations and noise. An elegant way to deal with these problems is to construct a hierarchical representation to handle different levels of detail, noise or abstraction. The idea of using a coarse-to-fine representation for graphs has its analogy in scale-space representations, like *maximally stable extremal regions* (MSER) [23] in images.

Eggert *et al.* [24] present the idea of *Scale Space Aspect graph* for 3D objects. The *Aspect graph* is a data structure that incorporates information about a series of two-dimensional views of the 3D object. This representation captures the structure of an object using different scale of details. This methodology is able to deal with object recognition at different resolutions instead of assuming infinitely high resolution images. This is specially important when some of the features cannot be found if the image is too small. Ulrich and Steger [25] propose to combine the idea of scale-space aspect graphs with the idea of similarity-based aspect graphs to develop a fast 3D object recognition approach.

Ahuja and Todorovic [26] present a region based approach for object recognition. Given an arbitrary image, they propose to apply a multi-scale segmentation algorithm to represent it using a hierarchical region graph (see Figure 4). This representation takes as nodes the set of regions that have been previously segmented. Two types of edges are considered, *lateral edges* that represent neighboring relations between regions and *ascendant-descendant edges* that capture the recursive embedding.

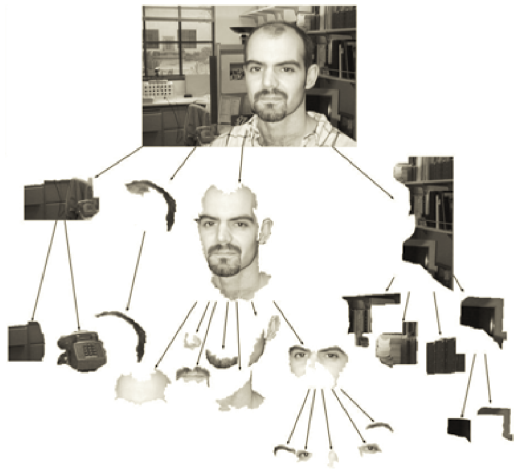


Fig. 4. Hierarchical graph representation proposed by Ahuja and Todorovic. Figure extracted from [26].

Broelemann *et al.* [27] propose a hierarchical graph representation for symbol spotting in graphical document images. The idea of the proposed hierarchy is to generate a graph which deals with the noise and distortion present in the vectorization. Therefore, this representation incorporates the typical vectorization errors such as merge nodes, remove dispensable nodes or merging a node with an edge. Finally, they face the problem of sub-graph matching in order to spot symbols which is solved by means of solving the maximal weighted clique problem in association graphs.

Very recently Mousavi *et al.* [28] have proposed a hierarchical representation close to the approach proposed in this work. They construct a hierarchical representation of the graphs and create an embedding combining the different levels of abstraction. Their work focuses on enriching their graph embedding with the information provided by hierarchy levels. The main difference constructing the hierarchy is that they decide the number of hierarchical levels to compute, therefore, they force the number of clusters for each level in a similar approach of *k-means*.

III. Hierarchical attributed Graph representation

This section describes the first contribution of this work. Namely, an attributed graph model with enriched labels to be able to deal with error-tolerant graph matching in large scale scenarios. The main properties of a desirable representation are the ability of summarizing the relevant features in a compact way so in a large scale retrieval problem non-promising graphs are retrieved in an efficient way (high recall), and at the same time the expressive capacity to represent the inherent variability among different instances of a visual object class. Therefore, comparing the underlying graphs extracted from images requires tolerance to high degrees of distortion and to capture different visual features by means of attribute vectors associated to nodes and edges. Afterwards, Section IV explains how to use these new representations to solve the problem.

Firstly, the construction of a hierarchical graph representation given an input graph is presented. Embedding approaches compactly capture the structural information of a graph into a feature vector. However, the structural information is partially lost. A graph reduction is a good option to encode the information in an abstract way but keeping the general structure of the original graph. A hierarchical cluster gathers together nodes that are likely to belong to the same unit. Moreover, reducing the number of nodes reduces drastically the time needed for graph matching.

Secondly, the local topological context of graph nodes is embedded into a vector that enriches the node attributes. This embedding is based on the *Morgan Index* [30] and computes the number of paths that are incident to each node. Therefore, it codifies the local connectivity or topology for every node. Also, a binarization of the node embeddings is proposed to take advantage of the properties of the binary vectors in terms of space and time. Binary codes are compact descriptors that capture the local context of an image key-point, according to a local neighborhood pattern, and represent it with a vector of bits. One of the most promising local descriptors is the efficient *Binary Robust Independent Elementary Features* (BRIEF) descriptor [29]. BRIEF is a binary descriptor that aims at quickly comparing local features while requiring few amounts of memory. The BRIEF descriptor outputs a set of bits obtained by comparing intensities of pairs of pixels within the local key-region. The good property of binary codes is that, since they are represented as vectors of bits, the comparison between two of them can be quickly computed with basic logical operations (usually XOR) using directly the CPU features.

A. Hierarchical graph construction

A hierarchical graph representation encodes information of the original graph at different levels of abstraction and detail. These different levels of the hierarchy allow to perform a matching at the desired level of abstraction. For instance, in a comparable way, the matching can be done at hand, arm and person level. Furthermore, if they do not belong to the same arm, they will not be the same hand. Let us formally define *hierarchical graph*,

Definition 3.1 (Hierarchical Graph): A *hierarchical graph* H is defined as a 6-tuple $H(V, E_N, E_H, L_V, L_{E_N}, L_{E_H})$ where V is the set of nodes; $E_N \subseteq V \times V$ are the neighborhood edges; $E_H \subseteq V \times V$ are the hierarchical edges; $\mathrm{L_V}$, $\mathrm{L_{E_N}}$ and $\mathrm{L_{E_H}}$ are three labeling functions defined as $\mathrm{L_V} : V \to \Sigma_V \times A_V^k$, $\mathrm{L_{E_N}} : E_N \to \Sigma_{E_N} \times A_{E_N}^l$ and $\mathrm{L_{E_H}} : E_H \to \Sigma_{E_H} \times A_{E_H}^m$, where Σ_V, Σ_{E_N} and Σ_{E_H} are two sets of symbolic labels for vertices and edges, A_V, A_{E_N} and A_{E_H} are two sets of attributes for vertices and edges, respectively, and $k, l, m \in \mathbb{N}$.

Two functions must be defined:

- *Contraction:* $c : G \to H$, defines the groups of nodes that must be gathered together. The contraction process can follow different criteria such as topology, features of the nodes or edges, etc.
- *Embedding:* $\varphi : G \to \mathbb{R}^n$, returns a vectorial representation of the subgraph that is contracted as an attribute. The embedding function can be seen as a signature of the subgraph that summarizes the information from one level to the other.

In this work, the contraction criteria will be based on the topology. A clustering process contracts the input graph iteratively from one level to the next one in the hierarchy. Let us study the proposed contraction criteria.

1) Hierarchy construction by community detection: In order to determine the group of nodes that are joined into one unique vertex, the Girvan-Newman algorithm [21] which is explained in Section II is applied. The output of this algorithm is a dendrogram providing a hierarchical clustering of the graphs nodes.

The proposed hierarchy encodes the graph into different levels of detail. For the contraction function, we propose to generate the nodes of the next level from at least two nodes of the current level. Therefore, the proposed methodology, defines a node as the centroid of the smallest cluster with at least two elements. Note, that the dendrogram given by the Girvan-Newman algorithm splits the nodes until each one belongs to a cluster with a single node. Therefore, the proposed contraction function uses the first level of each branch that does not leave any leaf alone. The idea is that each node of the hierarchy would contract a sub-graph and provide information about its topology. Hence, the corresponding nodes are contracted into only one vertex which is labeled with the result of the embedding function applied to these sub-graphs. Let us show an example using randomly generated graphs. Figure 5 shows three graphs that are used to illustrate the hierarchy construction. These graphs are created following the methodology that will be explained in Section V-A1. Figures 5a and 5c are graphs created with 15 nodes, and 3 clusters whereas Figure 5b is a distorted version of the first one.

Using these three graphs, Figure 6 shows the hierarchy that has been constructed with the proposed contraction function. We can observe the four levels of the hierarchy (in blue) for each graph. Moreover, between each hierarchy level, the nodes that are contracted are shown (in red). Notice that the hierarchy will always end up in a single node representing the whole graph.

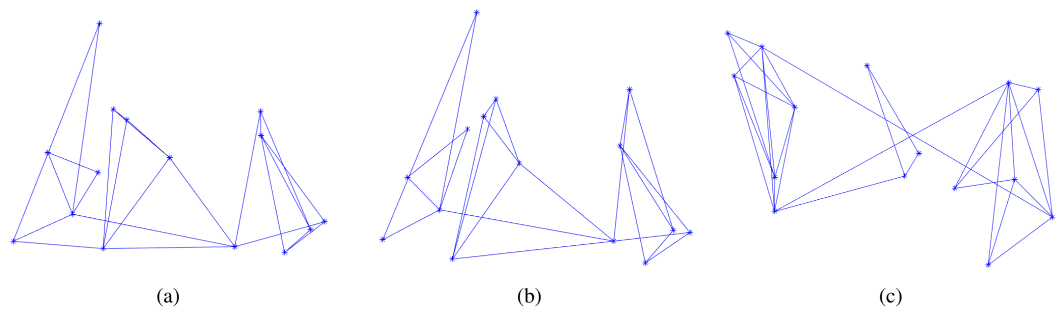


Fig. 5. Three graphs with 15 nodes, and 3 communities each (a) H_1, (b) H_2 (c) H_3.

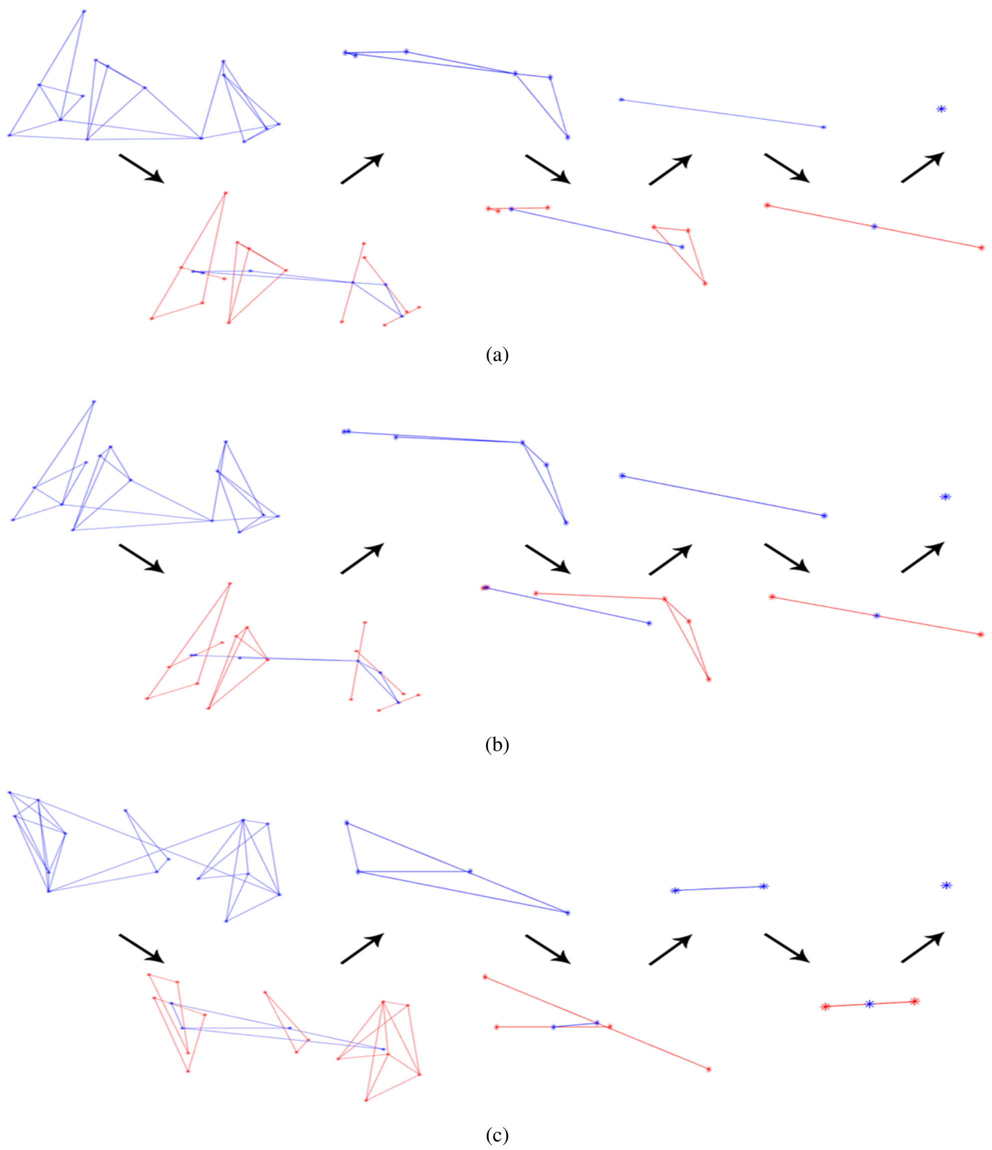


Fig. 6. Hierarchies for the graphs of Figure 5, (a) H_1, (b) H_2, (c) H_3.

2) Split articulation points: There are cases where slight deformations in the input graphs can lead to completely different hierarchies. Figure 7 shows a common subgraph that can lead to two possible hierarchies. In that figure, the possible hierarchies are presented. This ambiguity can lead to errors in the matching. Therefore, we propose to split the articulation points of the graphs.

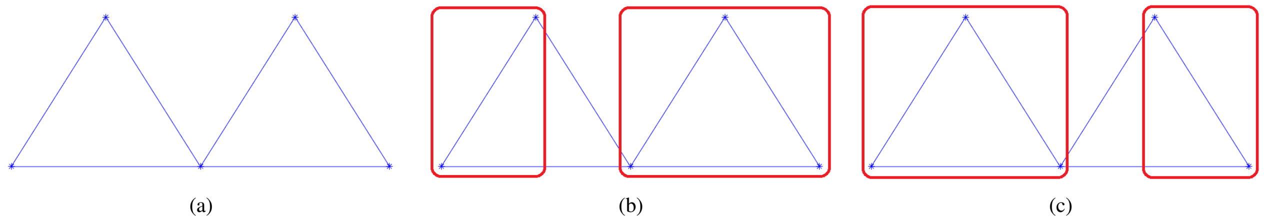


Fig. 7. (a) Ambiguity configuration that can significantly influence in the hierarchy construction, (b) and (c) in red two possible clusterings of nodes from the contraction function.

Definition 3.2 (Articulation Point): A vertex in an undirected graph is an articulation point if and only if removing it increases the number of connected components of the graph.

These nodes are of key importance, if they are classified in an incorrect cluster, they could change significantly the topology. The solution proposed in this work is to split the articulation point creating *virtual* nodes and disconnecting the graph. Therefore, it is stated that the articulation points divide and belong to two or more clusters. Figure 8 shows the splitting process following the previous example.

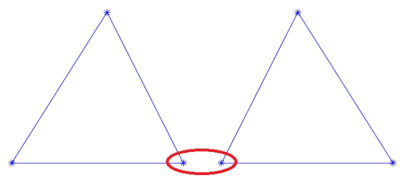


Fig. 8. Graph from Figure 7 where the articulation point has been overload generating a new *virtual* node leading to two communities.

Introducing this modification to the contraction function, the graphs presented in Figure 5 lead to a different hierarchy that is more stable. Figure 9 shows the construction of the different levels following the same color notation that has been explained. We can observe that the hierarchies are more stable between H_1 and H_2 taking into account the subgraph encoded in each node.

B. Node embedding formulation

This section presents a *Morgan Index* based node embedding. The node context is described in terms of the *Morgan index*, but it is enriched taking into account the labels of the neighboring nodes. Therefore, this embedding is proposed for discrete labeled graphs.

The *Morgan index* M of a graph G is a node feature, originally used to characterize chemical structures [30], that computes the node context in terms of its local neighborhood. This index is iteratively computed for each node $v \in V$ as follows:

$$\mathrm{M}(v,k) = \begin{cases} 1 & \text{if } k=0; \\ \sum_u \mathrm{M}(u,k-1) & \text{otherwise.} \end{cases}$$

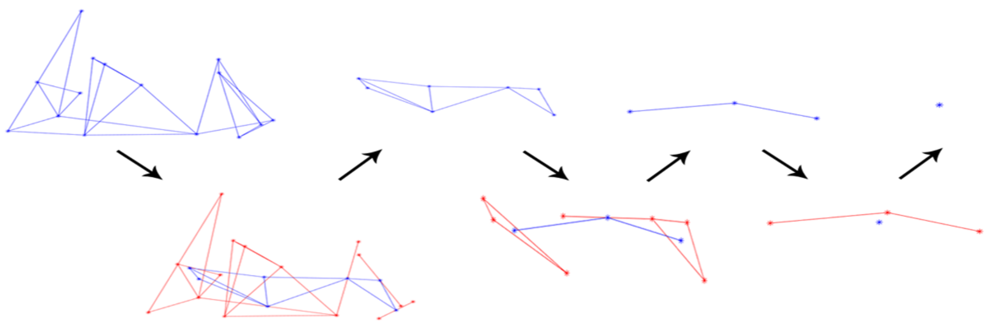
where u are the vertices adjacent to v. The Morgan index of order k associated to a given node v is defined as $M(v,k)$. It counts the number of paths of length k incident to node v and starting somewhere in the graph. The Morgan index can be computed using the values obtained from the exponentiation of the adjacency matrix A. An interesting property of the adjacency matrix A of any graph G is that the (i,j)-th entry of A^n denotes the number of walks of length n from the node v_j to the node v_i. Therefore, the Morgan index of order k from a node v_i is equivalent to the sum of the cells of the i-th row of the matrix A^k, formally:

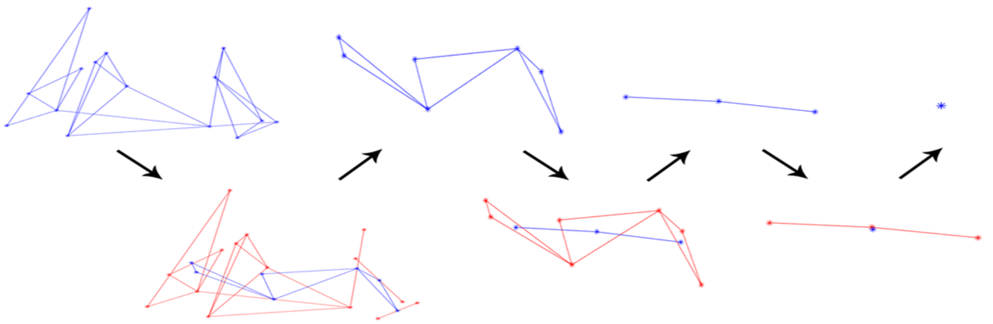
$$M(v_i,k) = \sum_j A^k(i,j),\ \ j = 1 \dots |V|.$$

Inspired by the topological node features proposed by Dahm *et al.* [31] we define the *local context* of a node v as a node embedding function computed in terms of the topological information of a sub-graph centered at v. Figure 10 shows the *local context* of a node, with $k=3$.

Let us denote as $M_l(v,k)$ the Morgan index of node v, order k and label l which counts the number of paths of length k incident at node v and starting at nodes labeled as l. According to this, the *local context* of a node v is formally defined as:

$$\nu(v) = [M_{l_1}(v,1), \dots, M_{l_1}(v,K), M_{l_2}(v,1), \dots, M_{l_2}(v,K), \dots, M_{l_{|\Sigma_V|}}(v,K)],$$


(a)


(b)

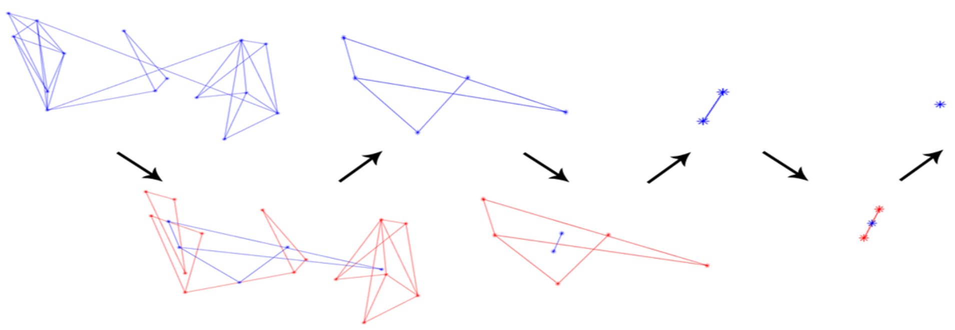

(c)

Fig. 9. Hierarchies for the graphs of Figure 5 splitting the articulation points, (a) H_1, (b) H_2, (c) H_3.

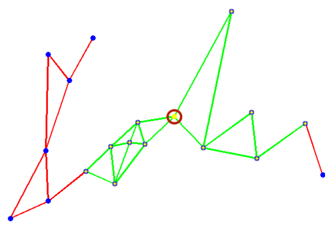


Fig. 10. *Local context* (in green) of length $k = 3$ for the node marked with a red circle.

where K is the maximum length of the paths incident in v that is considered. The value of K is dependent on each experimental set-up. Thus, every graph node is attributed by a $K \cdot |\Sigma_V|$ dimensional feature vector characterizing the number of paths incident at v of lengths up to K and starting at nodes for all the possible labels in Σ_V.

For indexation purpose, the context vector $\nu(v)$ is converted to a binary code $\hat{\nu}(v) = \{0,1\}^{K \cdot |\Sigma_V|}$ in terms of a list of corresponding threshold values T_i which are application dependent. Thus, instead of using $\nu(v)$, we can use this binary code to speed-up the indexation process with a small lose of information. Figure 11 illustrates the computation of the binary codes. In this example, we have used $\Sigma_V = \{A, B\}$ and $K = 3$, hence, the codes associated to nodes have length 6 ($|\Sigma_V| = 2$). The threshold value is set to the mean of each $M_l(v,k)$, therefore $T = [\frac{5}{4}, \frac{10}{4}, \frac{33}{4}, \frac{3}{4}, \frac{8}{4}, \frac{15}{4}]$.

Until now, we have defined vectors to encode information of the topology around a node v. However, the node v itself

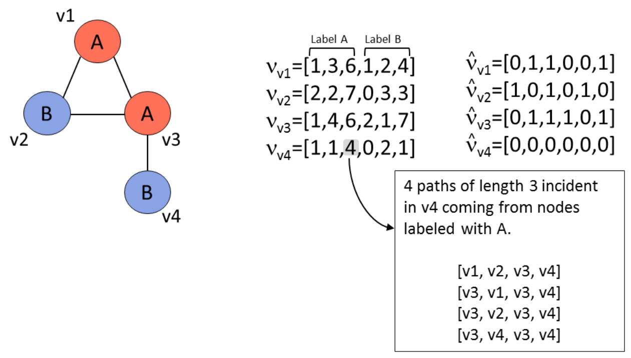


Fig. 11. Example of the node embedding computation from a labeled graph.

has not been codified. To solve this, we compute the paths of length 0. It can be interpreted as adding a flag to the vector indicating the label of the central node. Figure 12 shows the embedding computation from the graph of Figure 11 adding the label information of v.

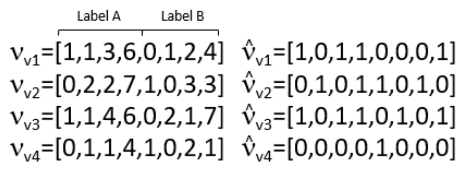

Label A
Label B
νv1=[1,1,3,6,0,1,2,4]
νv2=[0,2,2,7,1,0,3,3]
νv3=[1,1,4,6,0,2,1,7]
νv4=[0,1,1,4,1,0,2,1]
ν̂v1=[1,0,1,1,0,0,0,1]
ν̂v2=[0,1,0,1,1,0,1,0]
ν̂v3=[1,0,1,1,0,1,0,1]
ν̂v4=[0,0,0,0,1,0,0,0]

Fig. 12. Example of the embedding computation from the graph shown in Figure 11, in which we add paths of length 0.

The proposed embedding takes into account paths starting and ending in the same node. These paths can add some redundant information or noise. If we do not consider such cyclic paths, then we obtain a different embedding. Let us denote as $\tilde{M}_l(v,k)$ the Morgan index of node v, order k and label l which counts the number of paths of length k incident at node v and starting at nodes labeled as l but disregarding the ones starting at v. The modified definition is stated as follows:

$$\tilde{\nu}(v) = [\tilde{M}_{l_1}(v,1),\ldots,\tilde{M}_{l_1}(v,K),\tilde{M}_{l_2}(v,1),\ldots,\tilde{M}_{l_2}(v,K),\ldots,\tilde{M}_{l_{|\Sigma_V|}}(v,K)],$$

where K is the maximum length of the paths incident in v that is considered. Following the example of Figure 11, Figure 13 shows the new embedding. The different embedding variants will be compared in the experimental section.

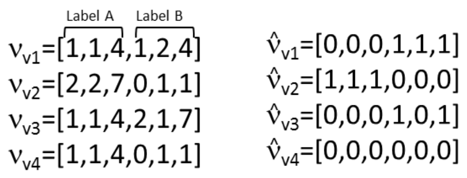

Label A
Label B
νv1=[1,1,4,1,2,4]
νv2=[2,2,7,0,1,1]
νv3=[1,1,4,2,1,7]
νv4=[1,1,4,0,1,1]
ν̂v1=[0,0,0,1,1,1]
ν̂v2=[1,1,1,0,0,0]
ν̂v3=[0,0,0,1,0,1]
ν̂v4=[0,0,0,0,0,0]

Fig. 13. Example of the node embedding computation from the graph shown in Figure 11, but disregarding cyclic paths.

IV. Error-tolerant graph matching and indexing

Given the graph based representations proposed in Section III, this section formalizes the matching and indexation approaches to exploit the power of the designed representations. Firstly, the simple graph matching used in this dissertation is explained. Afterwards, a coarse-to-fine matching to deal with the designed hierarchical representation is proposed. Finally, an indexation scheme for the binary node context embedding is defined.

A. Graph edit distance

As a baseline, the matching algorithm used to compare two graphs is the *bipartite graph matching* proposed by Riesen and Bunke in [8] and commented in Section II. *Bipartite graph matching* is a sub-optimal approximation of *graph edit distance* considering only local edge structure during the optimization process. This algorithm uses a cost matrix that codifies the edit costs, substitution, insertion and deletion between the source and target nodes. Once the cost matrix is defined, an edit operation must be assigned to each node minimizing the total cost. Let us define the cost matrix used for this approach.

Definition 4.1 (Cost Matrix): Let $G_1 = (V_1, E_1, L_{V_1}, L_{E_1})$ be the source and $G_2 = (V_2, E_2, L_{V_2}, L_{E_2})$ be the target graphs with $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$, respectively. The cost matrix C is defined as

$$C = \left[\begin{array}{cccc|cccc} c_{1,1} & c_{1,2} & \cdots & c_{1,m} & c_{1,\varepsilon} & \infty & \cdots & \infty \\ c_{2,1} & c_{2,2} & \cdots & c_{2,m} & \infty & c_{2,\varepsilon} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \infty \\ c_{n,1} & c_{n,2} & \cdots & c_{n,m} & \infty & \cdots & \infty & c_{n,\varepsilon} \\ \hline c_{\varepsilon,1} & \infty & \cdots & \infty & 0 & 0 & \cdots & 0 \\ \infty & c_{\varepsilon,2} & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \infty & \vdots & \ddots & \ddots & 0 \\ \infty & \cdots & \infty & c_{\varepsilon,m} & 0 & \cdots & 0 & 0 \end{array}\right]$$

where $c_{i,j}$ denotes the cost of a node substitution, $c_{i,\varepsilon}$ denotes the cost of a node deletion $c(u_i \to \varepsilon)$, and $c_{\varepsilon,j}$ denotes the cost of a node insertion $c(\varepsilon \to v_j)$.

Usually, the cost functions in the cost matrix are problem dependent. For this thesis, the cost functions are defined as follows:

Insertion and deletion costs. Both costs are equivalent (both can be seen as deletions in one graph or the other). Intuitively, the cost is computed in terms of the local configuration of the node defined by the incident edges and a constant corresponding to the node deletion. If the node is strongly connected the cost will be higher than for example a simple node that appears disconnected. Thus, the insertion cost has three terms:

$$c(\varepsilon \to v_j) = c(u_i \to \varepsilon) = \mathrm{we}_0 C_{\text{weightEdges}} + \mathrm{we}_1 C_{\text{edges}} + \mathrm{we}_2 t_{\text{vertices}}$$

where we_i are weighting factors; $C_{\text{weightEdges}}$ is the sum of the attributes of the edges incident in the node being deleted; C_{edges} is a measure of the density of the node computed as the ratio between the number of incident edges and the total number of graph nodes; t_{vertices} is a constant value that has to be set experimentally as a baseline cost for the node insertion or deletion.

Substitution costs. Computed in terms of the spatial position of the node, their labels or attributes and the similarity of the local structure.

$$c(u_i \to v_j) = \mathrm{wn}_0 D_{i,j} + \mathrm{wn}_1 C_{\text{attributes}} + \mathrm{wn}_2 C_{\text{localStructure}}$$

where wn_i are different weights. $D_{i,j}$ is the euclidean distance between the spatial position of the nodes u_i and v_j. This distance is normalized by the maximum node position of the both graphs. $C_{\text{attributes}}$ is the distance between the corresponding labels or attributes of the nodes. Finally, $C_{\text{localStructure}}$ is the edit operation cost on the incident edges.

Matching of the incident edges. In order to compute the edit cost on the adjacent edges ($C_{\text{localStructure}}$), the bipartite graph matching algorithm has been used again. Firstly a matrix of edit costs between the adjacent edges of both nodes is defined C_e with the same structure as C. In this case, the cost of edge insertion and deletion is a constant t_{edges}. The substitution costs are computed in terms of the edge attributes. i.e. weight, angle and length for the both edges to substitute.

$$c(e_i \to f_j) = \mathrm{we}_0 C_{\text{weightEdges}} + \mathrm{we}_1 C_{\text{angle}} + \mathrm{we}_2 C_{\text{length}}$$

where we_i are weighting factors, $C_{\text{weightEdges}}$ is the difference between the weight of the two edges; C_{angle} is the angle between them and C_{length} is equivalent to $1 - e_{\text{short}}/e_{\text{long}}$ where e_{short} denotes the length of the shorter edge and e_{long} the length of the longer one.

B. Coarse-to-fine matching

To take advantage of the hierarchical representation, we propose a coarse-to-fine graph matching approach. Let us denote H^i the graph representation at level $i = 1, \ldots, N$. It refines iteratively the matching starting in the most abstract or coarsest level (let us say $i = N$). The comparison is performed using the bipartite graph matching explained in Section IV-A taking the graph representation at level i without the hierarchical edges. If the distance at level i is small enough, the matching is performed at the next level $(i-1)$. The threshold to decide whether to advance in the hierarchy or not is application dependent and a threshold must be set experimentally. Starting the matching at the abstract level avoids a high number of comparisons at more detailed levels where the graphs are significantly bigger and consequently slower in matching time. Ideally, the last level is only used for graphs which are the ones you are searching or a very similar or related object. Moreover, the information about the matching level will be kept.

Table II and III show the resulting graph edit distance between the three graphs H_1, H_2 and H_3 (see Figure 5) using the proposed contraction functions. Based on the formulation of the graphs, H_1 and H_2 should have a small distance (H_2 is generated from H_1) whereas H_3 should give a higher one. Focusing in the first contraction function (see Table II), it is clear that we can easily notice which graphs belong to the same class for all the levels but 1. Here, a high distance between H_1 and H_2 is obtained because of the bad behavior of the proposed contraction function in some circumstances. Table III uses the splitting of the articulation points to stabilize the hierarchy solving the before mentioned problem. Now, there is a high dissimilarity between H_3 and the other two graphs for all the levels.

TABLE II
GRAPH EDIT DISTANCES FOR THE DIFFERENT LEVEL USING THE FIRST CONTRACTION FUNCTION. FROM LEFT TO RIGHT, FROM FINE TO COARSE RESPECTIVELY.

	Original			1st abstract			2on abstract			3rd abstract		
	H_1	H_2	H_3	H_1	H_2	H_3	H_1	H_2	H_3	H_1	H_2	H_3
H_1	0	0.0527	0.1753	0	0.0686	0.2404	0	0.2162	0.2594	0	0.0329	0.1301
H_2	0.0527	0	0.1855	0.0686	0	0.2502	0.2162	0	0.2693	0.0329	0	0.1524
H_3	0.1753	0.1855	0	0.2404	0.2502	0	0.2594	0.2693	0	0.1301	0.1524	0

TABLE III
GRAPH EDIT DISTANCES FOR THE DIFFERENT LEVEL USING THE GIRVAN-NEWMAN ALGORITHM OVERLOADING THE ARTICULATION POINTS. FROM LEFT TO RIGHT, FROM FINE TO COARSE RESPECTIVELY.

	Original			1st abstract			2on abstract			3rd abstract		
	H_1	H_2	H_3	H_1	H_2	H_3	H_1	H_2	H_3	H_1	H_2	H_3
H_1	0	0.0527	0.1753	0	0.0502	0.2331	0	0.0338	0.2940	0	0.0158	0.2335
H_2	0.0527	0	0.1855	0.0502	0	0.2353	0.0338	0	0.2940	0.0158	0	0.2281
H_3	0.1753	0.1855	0	0.2331	0.2353	0	0.2940	0.2940	0	0.2335	0.2281	0

C. Indexing

The previous section has explained how the hierarchy is used in the matching process. However, given a level of the hierarchy, the matching time can still be a problem. Therefore, given the node embedding presented in Section III-B a fast indexing scheme is constructed. For this approach, the binarized version of the proposed codes is used to construct the mentioned indexation scheme in terms of the Hamming distance. Figure 14 shows the pipeline that follows the proposed approach from the topological node labeling to the hash table.

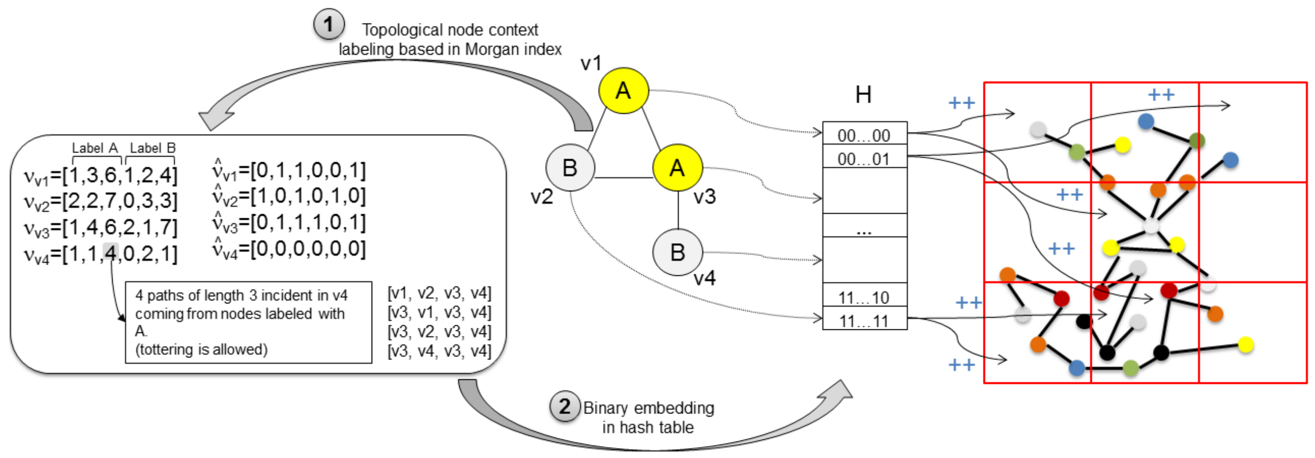


Fig. 14. Overview of the whole system. From the representation to the indexation.

The indexation proposed in this work is based on the concept of *focused graph retrieval* which can be defined as: Given a query graph G_q and a database of graphs $\{G_1, \dots, G_T\}$, a *focused graph retrieval* problem consists in finding the (sub)graphs of G_i similar to G_q. Thus, it is defined as finding inexact (sub)graph matchings between the query and the target graphs. In terms of a visual retrieval application, this process can be understood as not only retrieving the images of a database where a query object is likely to appear but finding the position in each retrieved image.

An inverted file indexing architecture in terms of node contexts is constructed. It stores a mapping from the binary topological features to the nodes of the target graphs in the database. This inverted file is therefore formulated as a lookup table $H : \{0,1\}^b \to \{v_i\}_{v_i \in V}$ that indexes a b-bit vector and returns a list of nodes whose context (binary code) is similar to the input code.

The last step is the actual (sub)graph matching process. With the indexing table H we only retrieve individual nodes, so it is necessary to implement a node consistency verification. With this purpose, we define a *partition* P of a graph G as a decomposition of it in n small (sub)graphs, $P(G) = \{g_1, \ldots, g_n\}$, where $g_i \subseteq G$. Hence, the lookup table H is reformulated as a hashing function that instead of returning nodes similar to the input binary code, returns (sub)graphs where these target nodes appear. Formally, given a query graph G_q and a database of graphs $\{G_1, \ldots, G_T\}$, for each node of the query graph $v \in V_q$, the indexation function H returns the (sub)graphs of the database, containing this vertex $H(v) = \{g_{q_1}, \ldots, g_{q_n}\}$, where g_q are n (sub)graphs of the target graphs $\{G_1, \ldots, G_T\}$ contained in the partitions $\{P(G_1), \ldots, P(G_T)\}$. The definition of the partition under which the database of graphs is decomposed in small graphs is application dependent. The (sub)graphs g_i can be seen as voting bins, according to a Hough-based principle. Thus, the final result consists of the (sub)graphs receiving a high number of votes.

Concerning the practical implementation of H, the similarity between binary codes is computed using the Hamming distance. The most straightforward solution is a brute-force linear scan, i.e. to compute the Hamming distance between the query vector and each vector in the database. The Hamming distance between two vectors consists in computing the XOR and counting the number of 1's in the resulting vector. This can be computed very fast on modern CPUs with logical operations, which are part of the instruction set. A fast hashing process like Locality Sensitive Hashing (LSH) [32] can be added to speed-up the indexation.

V. Experiments

This Section is devoted to validate the performance of the developed approaches. Three datasets are used illustrating different scenarios.

A. Datasets

Three datasets have been used. Firstly, a synthetic dataset to be able to understand the methodology and the hierarchy construction. Afterwards, an object image dataset is used to solve a classification problem and compare the performance against a state of the art technique. Finally, a historical handwritten documents dataset that has been represented using graphs is used to solve the word spotting problem.

1) Synthetic graph dataset: A synthetic graph dataset has been created in order to validate the proposed approach in a controlled scenario. The proposed graphs have been created using the idea of random graphs but introducing randomly a predefined number of clusters or communities. Afterwards, random deformations are applied to these graphs. Let us firstly define random graph following the Erdős-Rényi model proposed in [33].

Definition 5.1 (Random Graph): Given a positive integer n and a probability value $0 \leq p \leq 1$, define the graph $G(n, p)$ to be the undirected graph on n vertices whose edges are chosen as follows: for all pair of vertices v and w there is an edge (v, w) with probability p.

Firstly, 20 graphs were created, one for each class. The construction of the graphs consists of the following steps:

1) Set the number of nodes, n_v.
2) Set the number of clusters, n_c.
3) Randomly assign each node to a cluster, n_{v_i} where $i = 1, \ldots, n_c$.
4) Create a random graph with p_{in} probability of existence for each edge, $G(n_{v_i}, p_{in})$.
5) Define a random layout for each graph and separate the communities. The clusters should not be mixed.
6) Connect these graphs with p_{out} probability for each edge.
7) Assign a random value as node and edge attributes.

For our problem, we have used: $n_v = 14, 15, 16$, $n_c = 3, 4$, $p_{in} = 0.75$ and $p_{out} = 0.05$.

Afterwards, for each graph, deformations in the node position and node and edge attributes are applied. These deformations are applied following a normal distribution scaled to be within a predefined radius. The radius for the distortions has been set to 0.15, 0.20 and 0.25. Finally, with probability 0.01 two near nodes can be merged and any edge can appear or disappear. This kind of deformations are not meant to trick a *graph edit distance* algorithm but can help to understand the hierarchical process. Figure 15 shows some examples of the different classes and their corresponding hierarchy.

This dataset consists of 2000 graphs, 20 classes (100 graphs for each class) and 100 queries, 5 for each class randomly picked.

2) Columbia Object Image Library (COIL): The *Columbia Object Image Library* (COIL) [34] consists in 100 color images of objects against a black background. For each class, there are 72 images corresponding to different rotations of the object. Figure 16 shows one object of the dataset in six different positions.

Following the graph generation method used in [28], given an image, the underlying graph representation is extracted. The corners detected with the *Harris corner detector* [35] are used as the graphs nodes. Afterwards, the edges are generated using the Delaunay triangulation between these nodes. The final graphs do not have weights for the edges and only stores the coordinates of the corresponding nodes.

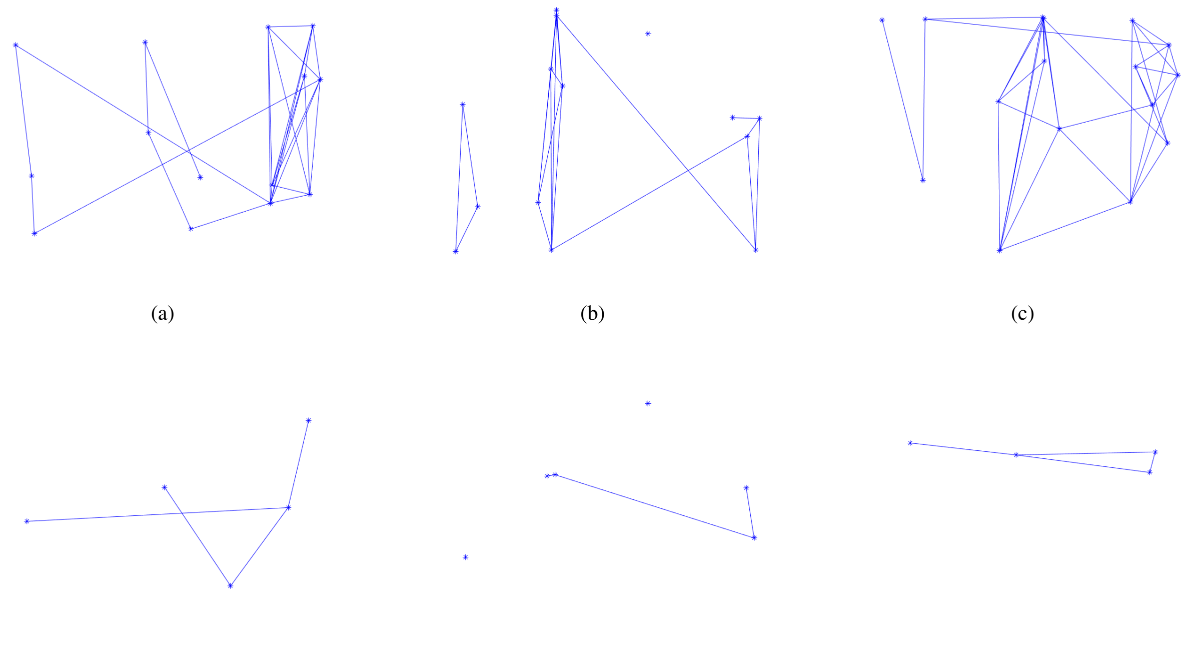

(a) (b) (c)
(d) (e) (f)

Fig. 15. (a), (b) and (c) Graphs corresponding to different classes in the synthetic dataset, (d), (e) and (f) are the corresponding hierarchy.

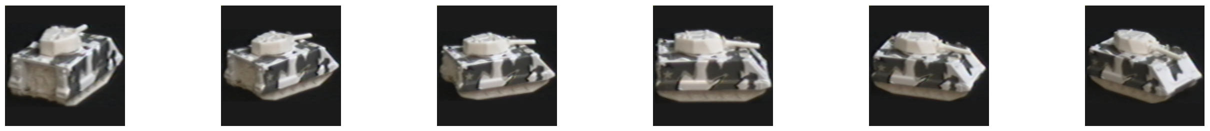


Fig. 16. Example of objects from the COIL-100 dataset [34].

For the experimentation, 15 classes are used, those with the maximum average number of nodes are selected. Moreover, [28] divide these graphs into three sets, training validation and test of 360, 75 and 150 graphs respectively. In the original paper, the process for obtaining this division is not reported. Hence, we have decided to generate the sets by randomly keeping the corresponding sizes.

3) BH2M: the Barcelona Historical Handwritten Marriages database: The Marriage Licenses Books conserved at the Archives of the Cathedral of Barcelona, called *Llibre d'Esposalles* is composed of 244 books with information of approximately 550,000 marriages celebrated in over 250 parishes in between 1451 and 1905. Each book was written by a different writer and contains information of the marriages during two years. The BH2M database [36] corresponds to the volume 69 written between 1617 and 1619, which contains 174 handwritten pages divided between training (100), validation (34) and test (40). For each page, the layout structure, transcription and semantic information is given. Figure 17 shows two pages from different centuries.

The structure of the strokes compounding handwritten characters is represented by attributed graphs where nodes correspond to basic primitives called *graphemes*. A *graheme* is the smallest unit used in describing the writing system of a language like loops, vertical lines, arcs, etc. Those graphemes are described using the *Blurred Shape Model* (BSM) descriptor [37]. All the descriptors extracted from the training set are used to create a *codebook*, therefore, each node will be labeled with a *codeword*. Edges represent adjacency relations between those primitives. This graph representation using *graphs of codewords* was proposed in [38], there, the graphemes are the convexities present in the handwritten words. Figure 18 shows an example of a word graph, firstly, the convexities are computed (see Figure 18a) and secondly, the graph is generated (see Figure 18b).

The motivation for representing handwritten words using graph-based methods is to keep information of the two dimensional structure of a handwritten text. This representation aims to avoid the loss of information provided by the appearance-based representation in one dimensional scalar vector of features. The nature of handwriting suggests that the structure is more stable than the pure appearance of the handwritten strokes. This is specially important when dealing with the elastic deformations present in different handwriting styles.

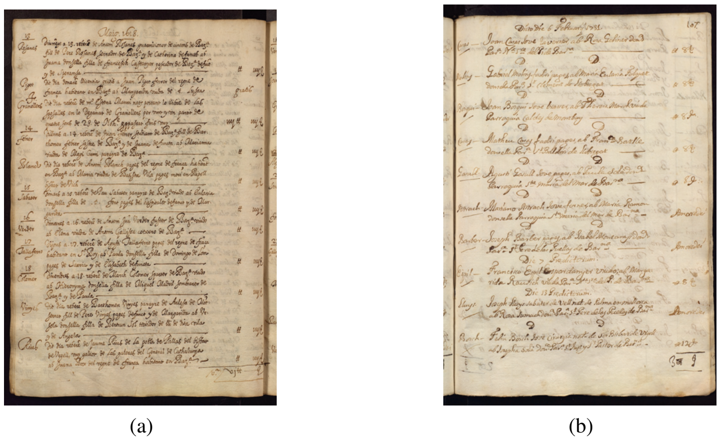


Fig. 17. Examples of pages from different centuries of the marriage records. (a) 1618, volume 69 (b) 1729, volume 127.

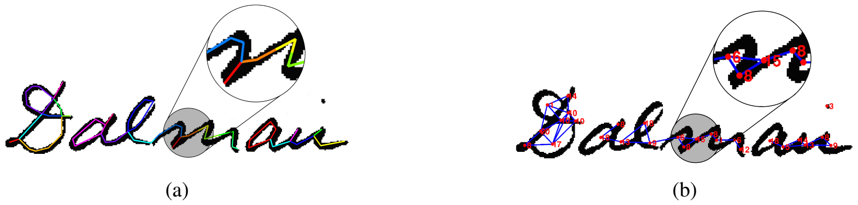


Fig. 18. Construction of the graph representation for a word, (a) convexities from the word vectorization, (b) final grapheme graph.

B. Evaluation

The evaluation proposed in this work uses the classic metrics in the context of information retrieval scenarios [39]. Let *ret* be the set of retrieved elements from the dataset and *rel* be the set of relevant objects with regard to the query. Let *True Positive* (*TP*) be the set of (correctly) relevant retrieved elements ($|\text{ret} \cap \text{rel}|$), *False Positive* (*FP*) be the set of incorrectly retrieved elements ($|\text{ret} \cap \overline{\text{rel}}|$), *True Negative* (*TN*) be the non relevant elements that have not been retrieved ($|\overline{\text{ret}} \cap \overline{\text{rel}}|$) and *False Negative* (*FN*) be the relevant elements that have not been retrieved ($|\overline{\text{ret}} \cap \text{rel}|$). In this work the metrics used to evaluate the performance, are:

- **Precision:** Is the probability that a (randomly selected) retrieved element is relevant. Precision is defined as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Alternatively, it can be defined as:

$$\text{Precision} = \frac{|\text{ret} \cap \text{rel}|}{|\text{ret}|}$$

Let *P@n* be the precision at *n*, which is obtained by computing the precision at a given cut-off rank, considering only the n top-most results returned by the system.

- **Recall:** Is the probability that a (randomly selected) relevant element is retrieved. Recall is defined as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Alternatively, it can be defined as:

$$\text{Recall} = \frac{|\text{ret} \cap \text{rel}|}{|\text{rel}|}$$

- **Specificity:** Is the probability that a non relevant element is identified. Specificity is defined as:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Alternatively, it can be defined

$$\text{Specificity} = \frac{|\overline{\text{ret}} \cap \overline{\text{rel}}|}{|\overline{\text{ret}}|}$$

- **Mean Average Precision (mAP):** Is computed using each *precision* value after truncating at each relevant item in the ranked list. For a given query, let $r(n)$ be a binary function on the relevance of the n-th item in the returned ranked list. Firstly, let us define *Average Precision* (*AP*):

$$\text{AP} = \frac{\sum_{n=1}^{|\text{ret}|} P@n \times r(n)}{|\text{rel}|}$$

Taking the *AP* definition, the *mAP* is defined as:

$$\text{mAP} = \frac{\sum_{q=1}^{Q} \text{AP}(q)}{Q},$$

where Q is the number of queries.

C. Synthetic dataset: Graph hierarchy

The first experiment has been designed to exhaustively evaluate the performance of the hierarchical graph representation, we have generated the synthetic dataset explained in Section V-A1. This dataset was specially created to test the proposed hierarchy in a controlled scenario. Hence, the kind of deformations applied are easily recovered by *graph edit distance* algorithms which perform a *mAP* of 100%.

It is interesting to understand the representational power of the hierarchy. As an embedding function, to codify the topology of nodes that are contracted in the hierarchy, the mean of node labels, and the mean of the Morgan Index from order 0 to 2 are concatenated into a vector. Figure 19 shows the precision-recall curve using the first level of the hierarchy for both contraction functions. These curves give a *mAP* of 83.01 and 79.66 for the Girvan-Newman based contraction function and the modification splitting the articulation points respectively. Note that the proposed modification was meant to stabilize graphs with some particular behaviors.

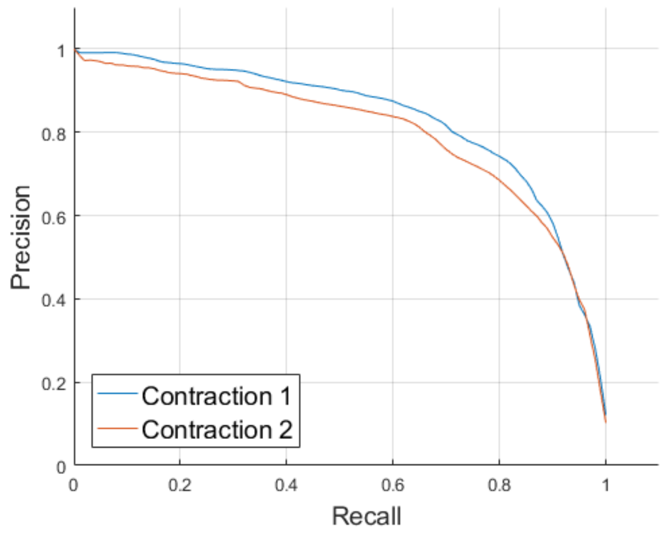


Fig. 19. Precision-Recall curve for the synthetic dataset in the first level of hierarchy. *Contraction 1*: Hierarchy using the Girvan-Newman based contraction function, with $mAP = 83.01\%$. *Contraction 2*: Splitting articulation points, with $mAP = 79.66\%$.

As it has been explained in Section IV, the proposed coarse-to-fine matching aims to increase the time performance with a small loses on the matching metrics. Figure 20 shows how the *mAP* changes depending on the threshold used to advance in the hierarchy during the matching and the percentage of comparisons that have been avoided. In this case, only two hierarchical levels (those with more information) are used. These plots allow to set the threshold that gives a good trade-off between the metric and the percentage of comparisons that are avoided in the fine level. Notice that the *mAP* increases faster than the percentage of comparisons that must be done in the fine level. The threshold has been set to 0.225 to obtain a good trade-off between both metrics. Figure 21 shows the performance using two levels with the selected threshold. Note that almost 70% of the comparisons can be avoided only losing a 10% of the *mAP*.

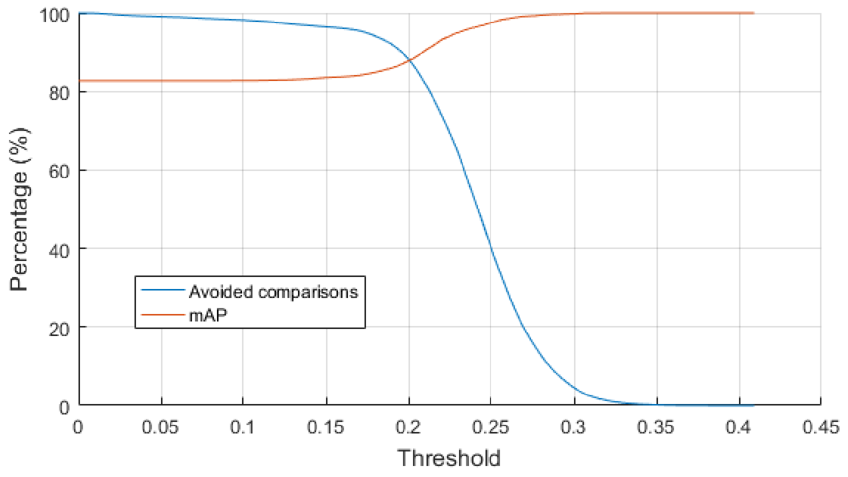


Fig. 20. Comparison between the percentage of *avoided comparisons* and the *mean average precision* changing the threshold for the first contraction function.

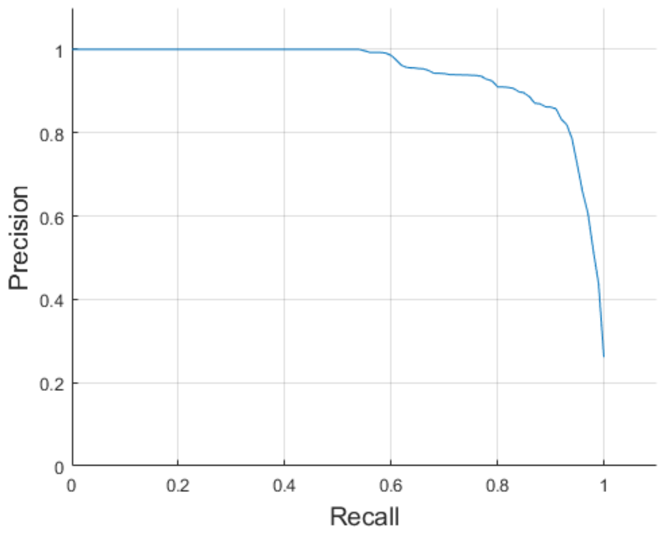


Fig. 21. Precision-Recall curve for the synthetic dataset using the coarse-to-fine matching with two levels. $mAP = 94.12\%$ with a threshold 0.225, which avoids 69.27% comparisons.

The effect of the hierarchy in terms of time[3] is reported in Table IV. Table IV compares the time in seconds for the original graphs, the first level of the hierarchy and finally the matching using both levels. Note that there is an speed up of 2.90 in terms of time.

TABLE IV
TIME PERFORMANCE COMPARISON IN TERMS OF TIME AND MAP.

Approach	**Time** (s)	**mAP** (%)
Original graph	390.59	100
Hierarchy only	16.97	83.01
Coarse-to-fine (2 **levels**)	134.51	94.12

D. Columbia Object Image Library (COIL): Graph hierarchy

This experiment for the hierarchical graph representation reproduces the one proposed by Mousavi *et al.* in [28]. The objective of this experiment is to classify color images into their corresponding classes using their graph representation. To demonstrate the representational power of the proposed framework, a *k-Nearest Neighbor* (k-NN) classifier is used. As k-NN is a very

[3]It is assumed that the hierarchy is computed off-line.

simple classifier, it shows the quality of the graph representation. For this experiment, the original graphs and the 1st and 2nd abstract levels are evaluated. As it has been done in the previous sections, to compute the distances between graphs, the bipartite graph matching [8] has been applied. Using the validation set, all the parameters to compute the graph edit distance have been set. As it has been explained, the graphs from this dataset are constructed using Delaunay triangulation. Hence, the obtained graphs do not have articulation points and both contraction functions lead to the same hierarchy. As an embedding function, the mean of the Morgan Index of length 1 and 2 are concatenated.

Table V shows the comparison with [28] in terms of classification accuracy. It is observed that the original graphs do not achieve the same accuracy provided by this paper. This can be produced because of the set division that has been done randomly due to the lack of information provided in the paper. Note also the loss in accuracy for the 1st and 2nd abstract levels in comparison to Mousavi *et al.* approach. However, it is important to remark that the proposed methodology reduces the size of graphs with respect to the other approach.

TABLE V
THE CLASSIFICATION ACCURACIES (%) OF K-NN IN GRAPH DOMAIN ON THREE ABSTRACT LEVELS OF COIL DATASET.

k-NN	**Accuracy of [28]** (%)			**Accuracy of the proposed approach** (%)		
	Original	1st **abstract**	2nd **abstract**	**Original**	1st **abstract**	2nd **abstract**
$k = 1$	100.00	98.17	87.00	96.67	85.33	43.33
$k = 3$	97.00	94.83	81.67	94.00	80.67	44.00
$k = 5$	90.00	88.83	78.17	92.00	84.67	51.33
Time (s)	-	-	-	4701.08	348.41	25.44

Finally, Table VI shows the reduction in terms of number of nodes and edges for each level. Observe that the number of nodes and edges is in average divided by 3 for each level in the hierarchy. Notice also that the original dataset used in [28] has a totally different graph size although the procedure explained in the paper has been followed. They do not provide the reduction in terms of graphs size for their hierarchical approach.

TABLE VI
SIZE OF THE GRAPHS FOR THE COIL DATASET FOR THE THREE ABSTRACT LEVELS.

	Mousavi *et al.* [28]		**Proposed approach**	
	$\|V\|$ **(min,max,avg)**	$\|E\|$ **(min,max,avg)**	$\|V\|$ **(min,max,avg)**	$\|E\|$ **(min,max,avg)**
Original	(18 , 79 , 42.60)	(45 , 228 , 116.10)	(22 , 122 , 66.25)	(54 , 358 , 185.39)
1st **abstract**	n/a	n/a	(7 , 45 , 22.31)	(13 , 128 , 56.23)
2nd **abstract**	n/a	n/a	(2 , 17 , 7.39)	(1 , 43 , 14.06)

E. Handwritten word spotting: Baseline

The present experiment has been designed to evaluate a graph-based methodology in a real application case. It belongs to the area of handwriting recognition, in particular, word spotting in the context of the preservation of historical manuscripts stored in archives, libraries and museums. Once large amounts of documents are digitized, the challenge is the extraction of information for consultation purposes. A full transcription using handwriting recognition techniques is not feasible nowadays because of the variability of the text styles, the bad physical preservation of the sources, and because handwriting recognition techniques require large amounts of annotated images to train, which is not always available.

Word spotting is an alternative for content indexing. Word spotting is the task of retrieving the instances of a given query word. It is usually formulated as a visual object detection problem, where the query word and the image words are represented by features invariant to visual distortions. Most word spotting techniques use statistical representations (e.g. HOG, SIFT) of the word images, e.g. [40] and [41]. However, there are also few approaches using structural representations [42], [43].

The experimentation proposed in [38] is extended to the whole database with a more accurate study of the number of clusters to create the codebook. There, a query by example segmentation-based word spotting framework is validated using a subset of the dataset using only 514 queries and 6544 segmented words. The retrieved graphs are obtained using the graph edit distance given by the bipartite graph matching approximation [8]. The graph edit distance is computed using the same parameters validated in [38]. Moreover, two distances are tested between the codewords, L_1 and L_2 distances. The proposed experiment is equivalent to perform the matching using only the finest level of the hierarchical representation.

Firstly, Table VII illustrates the influence of the size of the codebook for node labels in the retrieval performance. The performance slightly increases with the number of clusters that determine the codebook of node labels ($\sim 1\%$). However, changing the distance (L_1 and L_2) between clusters gives almost the same performance ($\sim 0.07\%$). Thus, the number of clusters used to generate the codebook has a little influence on the overall performance and the discriminative information is achieved even with a codebook of size 20. Figure 22 shows the *Precision-Recall* curve of the best configuration using the test set: codebook size of 200 and L_2 distance between codewords.

TABLE VII
CODEBOOK SIZE AND DISTANCE STUDY BETWEEN CODEWORDS IN BH2M DATASET USING THE VALIDATION SET.

	mAP (%)	
# Clusters \Codebook distance	L_1**-distance**	L_2**-distance**
20	64.73	64.71
50	65.31	65.35
100	65.74	65.77
200	65.79	**65.79**

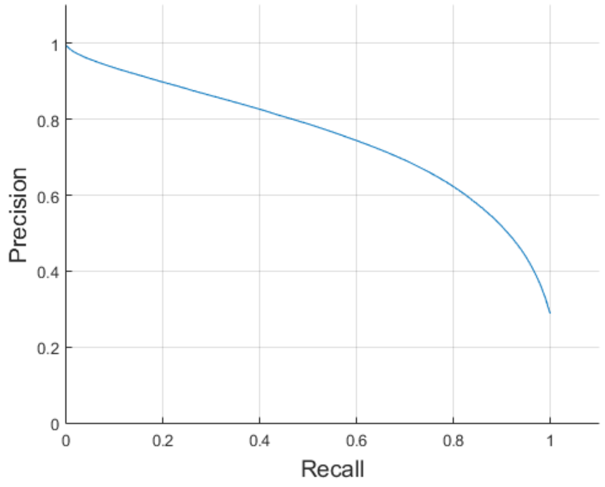


Fig. 22. Precision-Recall curve with 200 clusters and L_2-distance in the test set. $mAP = 70.74\%$.

In the comparison of statistical versus structural representations for handwritten words reported in [42], the main disadvantages of structural approaches are the time complexity and scalability to large collections. Table VIII shows a comparison between this graph-based methodology against other word spotting techniques based on statistical approaches. Note that it is not directly comparable with the last two methods because they are segmentation-free techniques. Moreover, the last approach only uses a subset of the database with only a few transcriptions. However, from the figures of the table, we can see that in terms of the application, graph-based models although being competitive regarding the state of the art, do not reach the same performance than methods specifically designed. For more comparisons, the interested reader can refer to [38].

TABLE VIII
COMPARISON WITH STATISTICAL TECHNIQUES REPORTED IN THE LITERATURE.

Approach	**mAP** (%)
This approach (200 Clusters)	70.74
DTW + Vinciarelli [44]	31.51
HOG+EWS [45][1]	51.35
BoVW + SIFT [41][1][2]	90.17

As it has been stated in Section II-B, graph kernels and embeddings can be used to deal with graphs without an explicit matching. Comparing graph edit distance methods (bipartite graph matching) with typical graph kernels, we can notice a big loss of information (mainly in the structure) for this dataset. To perform this comparison, the *labeled random walk* kernel from [46] and the *labeled graphlet kernel* from [47] have been chosen. The first one counts the number of common walks between both graphs and the second one computes a histogram for all possible connected graphlets of length three. Table IX shows a comparison in terms of *mAP* between these approaches. For this comparison, only 20 clusters have been used as codebook size because the kernel computation has polynomial complexity in terms of the node labels. For instance, the histogram computed by the *labeled graphlet kernel* is of size 16,000 using 20 labels whereas, it is 16,000,000 using 200.

The loss of performance between a graph edit distance algorithm against graph kernel functions is produced by two main factors. Firstly, because we are using small graphs that are connected in similar topologies, mainly triangles. Secondly, for the graph edit distance computation, the codeword distance is computed through the codebook centroid whereas, in the kernel computation, the labels are discretized. Hence, for the computation of walks and graphlets, all the labels have the same distance.

[1]Segmentation free.
[2]Subset of BH2M dataset with only 21 different transcriptions.

TABLE IX
COMPARISON WITH GRAPH MATCHING TECHNIQUES GIVEN THE SAME REPRESENTATION.

Approach	**mAP** (%)
This approach (20 Clusters)	69.45
Labeled Random Walk kernel [46]	9.08
Labeled Graphlet kernel [47]	9.88

Therefore, having a node classified into a different cluster introduces more noise in the kernel computation than in the graph edit distance.

F. Handwritten word spotting: Graph hierarchy

This experiment is a continuation of the previous one. It has been designed to evaluate the hierarchical representation for word spotting purposes. Therefore, another level of the hierarchy is introduced to the matching process. Figure 23 shows a word and the corresponding hierarchy using the two proposed contraction functions. Qualitatively, splitting the articulation points is useful to make the hierarchy more stable for the kind of graphs appearing in this dataset.

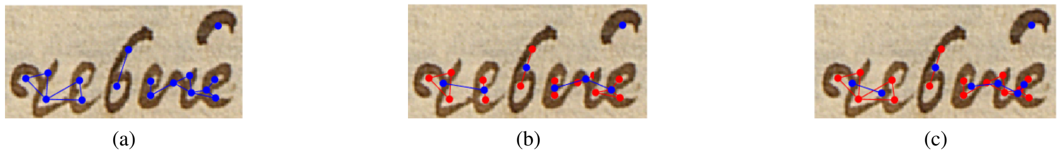


Fig. 23. Example of hierarchy construction for the BH2M dataset, in blue the corresponding graph and in red the vertices that are contracted, (a) input graph, (b) hierarchy for the Girvan-Newman based contraction function, (c) hierarchy for the Girvan-Newman based contraction function splitting the articulation points

As it has been explained, each node of the graph representation corresponds to an extracted unit or grapheme of the original word. Figure 24 demonstrate the hierarchy that is created in terms of image units (graphemes) given the word *Dalmau*. For each level,it can be observed how the graphemes are combined to generate parts of the letters, letters and at the end, a word. Also, the graphs that are generated for each level are shown.

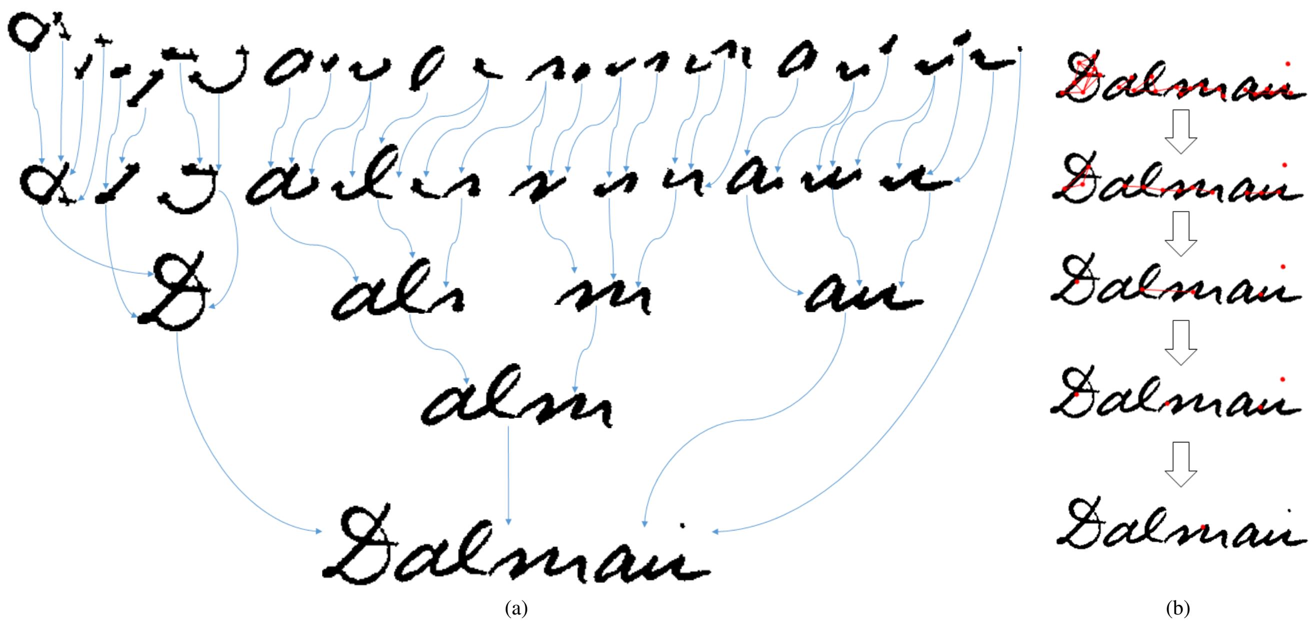


Fig. 24. Examples of Hierarchy tree (a) Shows the decomposition in graphemes of the original word and how they are joined following the contraction function, (b) Shows the graphs corresponding to each level for the same word.

Firstly, the proposed hierarchical representation is studied using the validation set. The *embedding function* used for this dataset is a vector that counts the number of paths of length up to k from any node to a node with label i. However, the best option has been proved to be $k = 0$ that is a vector counting the number of nodes with label i (similar to a *bag of words*). Reproducing the first experiment (see Table VII) but changing the graphs by the second level of the hierarchy, leads to a loss of performance ($\sim 20\%$). This loss is very similar to the obtained in the previous experiment. Table X show the comparison between the contraction functions and the original graphs. As expected, overloading the articulation points makes the graphs more stable.

TABLE X
STUDY OF THE CONTRACTION FUNCTION IN THE FIRST LEVEL OF THE HIERARCHY AND NUMBER OF CLUSTERS FOR GRAPH LABELS (VALIDATION SET).

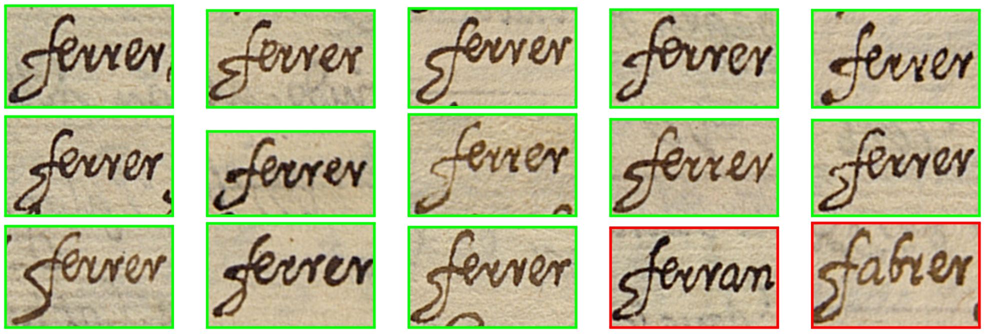
	mAP (%)		
# Clusters \Contraction function	**Original**	**Girvan-Newman based**	**Splitting articulation points**
20	64.71	34.36	**44.16**
50	65.35	33.10	43.49
100	65.77	30.89	41.48
200	65.72	27.57	38.76

Table XI shows qualitative results for a given query *ferrer*. The results are provided either for the original graphs and for the first abstract level using the contraction function that splits the articulation points. Note that the incorrect results for both graph representations belong to words that are similar in terms of the shape.

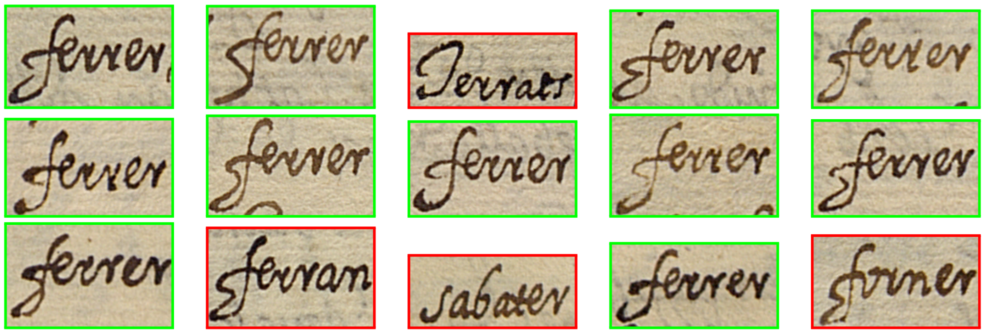
TABLE XI
QUALITATIVE RESULTS FOR THE QUERY *ferrer* USING THE ORIGINAL AND THE FIRST LEVEL OF THE HIERARCHY. GREEN WORDS ARE THE CORRECTLY RETRIEVED WHEREAS, THE RED ONES ARE INCORRECTLY RETRIEVED

Query:

Original:



1st abstract:



The first two levels of the hierarchy are combined using the proposed coarse-to-fine matching. These levels are the ones providing more information. As it has been explained, the hierarchy levels are not used alone but combining them to achieve the desired trade-off between speed and performance. Combining both levels means to decide the distance required at the abstract level to be worth to perform a more costly graph comparison. The performance combining those two levels depending on the threshold is studied. Figure 25 shows the evolution of the *avoided comparisons* and *mAP* with the threshold. This figure makes clear that many comparisons can be avoided before having a big loss of performance.

Now that the data behavior has been studied, let us fix a threshold to get the results in the test set. The used thresholds have been decided using Figure 25 to show the effect of the threshold both in terms of performance and time. Table XV shows a

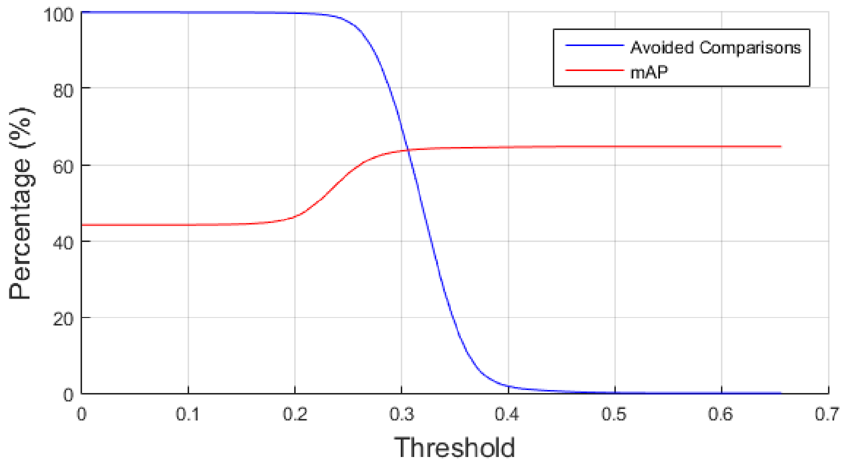


Fig. 25. *Avoided comparisons* and *mAP* evolution changing the threshold to decide whether or not use the second level of the hierarchy.

comparison against the original methodology and the indexation approach both in terms of time and performance. Recall tells us how many correct graphs are identified using the abstract level and therefore matched using the original graph. Besides, specificity points out the number of incorrect graphs that are detected thanks to the hierarchy. Notice that using 0.25 as the threshold, a speed up of 5.02 is obtained losing only the 30% of the correct words because of the hierarchy.

G. Handwritten word spotting: Graph node indexation

The experimental case for the proposed node embedding consists in using these vectors in an indexing scheme to retrieve instances of a given query word from the BH2M dataset presented in Section V-A3. For the sake of simplicity, only the original graphs are used to validate this indexation framework. For the proposed dataset, using the indexation approach in other levels of the hierarchy is not feasible because the graphs are reduced too much to be able to obtain discriminative information in terms of the local topology. The partition of the target graphs consists of the subgraphs corresponding to the segmented words, therefore, all the results provided for the different methodologies proposed in this work are comparable. Thus, the votes are accumulated for each image containing words likely to be the query one. Then, a more accurate search can be performed in the images that have received more votes. The true positives are those regions that contain the correct word and have at least a minimum amount of votes. Figure 26 shows qualitatively the behavior of the index scheme using a word graph as a query and a whole page to search it. It is observed, that the votes are mainly focused on a few words rather than in the whole page. Hence, the comparison can be done in those specific regions.

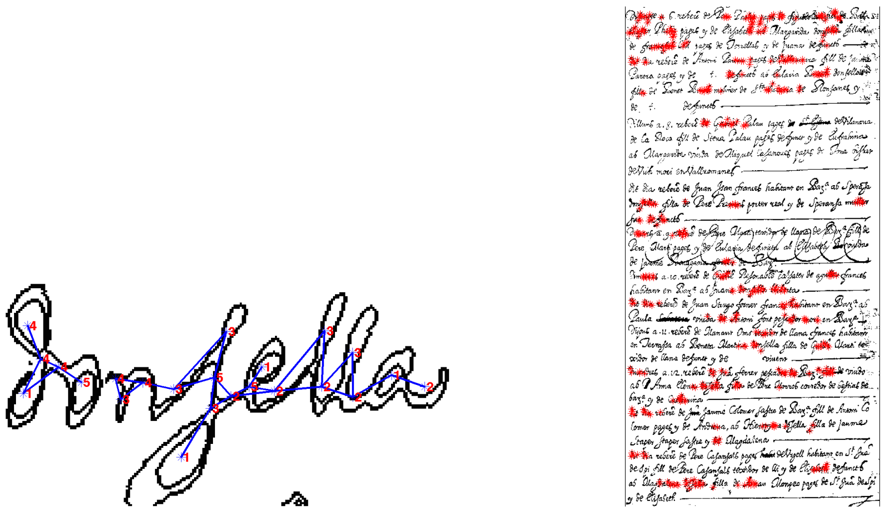

(a) (b)

Fig. 26. Qualitative results (a) a query word and its corresponding graph (b) A full page and the locations where query nodes are detected.

The proposed methodology aims to maximize the *recall* whereas keeping a good *specificity*. It is not focused on the *precision* but on the rejection of incorrect words in a fast way. The idea is to index the relevant elements (*recall*) but remove as many non-relevant elements as possible (*specificity*).

For all the experiments in this section, one node is considered to receive a vote if and only if the Hamming distance between its binary local context based on *Morgan Index* and the one from the query is less than 10 (set experimentally). Moreover, to decide whether or not a word has enough votes, the threshold is adapted depending on the amount of nodes in the query graph. It is not the same to receive two votes from a small graph than two from a large one. Let us define this value as $|V_q| \cdot x$ where $x \in (0, 1]$ is a parameter that depends on the application and should be set using the validation set, and V_q is the set of vertices of the query graph. Firstly, 20 is used as the number of clusters used to generate the codebook and the four possible embeddings that have been presented in Section III-B and the local context size k are evaluated. Afterwards, the study is extended to the other number of clusters presented in Table VII.

Let us denote *LC* as the *local context* (as it has been illustrated in Figure 10), which corresponds to paths of length up to k in the node embedding computation. In addition, *NF* which states for *node flag*, a binary attribute to indicate whether 0-length paths are considered or not. Finally, *AC* for *avoid counts* is used to indicate whether cyclic paths are disregarded or not.

The first parameter to tune is the order k for the *LC*. Let us set *NF* and *AC* to 0 in order to find the best k. Three different values has been studied: 2, 3, and 4; and for each one, the *recall* and *specificity* are taken into account. Figure 27 shows the evolution of both metrics depending on the threshold considered for the indexation. From these plots, it is difficult to interpret which is the best option. Notice that all the cases have a similar behavior with respect to recall and precision. Therefore, we fix the desired recall and see the corresponding specificity for each case.

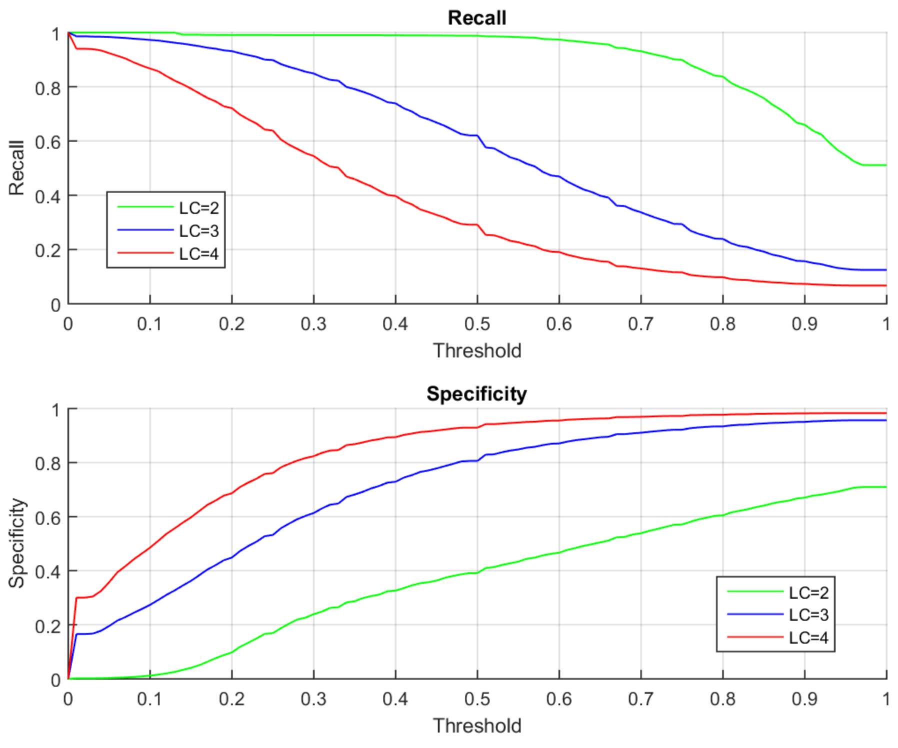


Fig. 27. Recall and specificity evolution depending on the threshold for the *local context* (LC): 2, 3 and 4.

Table XII shows the specificity for each *local context* fixing the recall. The recall used for this table is an approximation due to the threshold sampling. This is a summary of Figure 27 that makes clear that the best option is to set LC to 3. There we can observe that the *specificity* is better for both recalls than the other local context sizes. As we are using a dataset with quite small graphs, using a big local context introduces noise and redundancy whereas a small one, cannot encode enough information about its topology.

Until now, we have studied the influence of the parameter *LC*. Let us study the combinations of *NF* and *AC* with the fixed *LC* to 3. Similarly to Figure 27, Figure 28 shows the evolution of *recall* and *specificity* metrics for the four possible embeddings. All the curves have a similar behavior, therefore it is difficult to state which is the best embedding. Observe that adding *NF* increase the embedding vector size. Therefore, the vector representation becomes more sparse and the distance between them increase.

As it has been shown before, Table XIII shows the specificity for each methodology at a fixed recall. Notice that introducing

TABLE XII
SPECIFICITY FOR A FIXED RECALL (APPROX.) AND *local context* (LC)

Configuration	Specificity	
LC	Recall $= 0.8$	Recall $= 0.7$
2	0.62	0.66
3	**0.67**	**0.75**
4	0.59	0.71

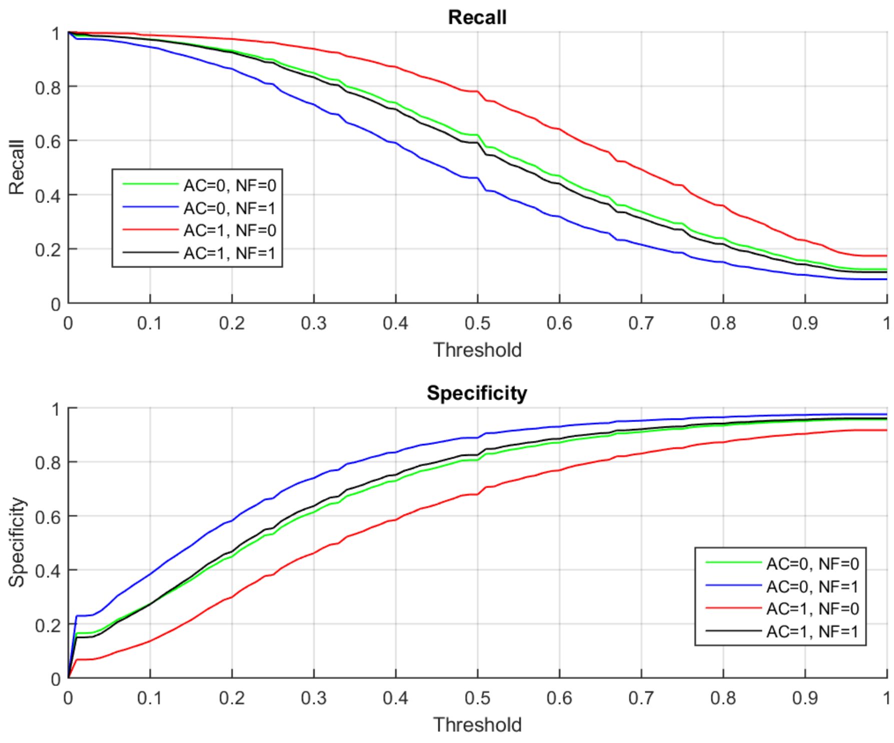


Fig. 28. Recall and specificity evolution depending on the threshold for the combination of *NF* and *AC*.

the proposed modifications, the performance does not seem to increase significantly and it is only noticeable for a recall of 0.7 ($\sim 2\%$). Using the proposed table, the most stable embedding correspond to $LC = 3$, $AC = 1$ and $NF = 1$.

TABLE XIII
SPECIFICITY FOR A FIXED RECALL (APPROX.), CHANGING *AC* AND *NF*

Configuration		Specificity	
AC	**NF**	Recall $= 0.8$	Recall $= 0.7$
0	0	**0.67**	0.75
0	1	0.66	**0.77**
1	0	0.66	0.73
1	1	**0.67**	**0.77**

Now, let us study the effect of the node labels for 20, 50, 100 and 200. Intuitively, having a larger codebook will make our system to be more restrictive. Hence, the recall is expected to decrease faster as more labels are used. Figure 29 shows the former mentioned behavior.

Table XIV demonstrates that for the given parameters, the best number of clusters to generate the node labels is 20. This table shows that fixing the desired precision, the specificity fall with the number of clusters. As it has been explained, the embedding size can be computed by $K \cdot |\Sigma_V|$, in this case, $(K+1) \cdot |\Sigma_V|$ because of the option *NF*. Therefore the vector size is 80 using 20 clusters whereas it is 800 using 200. Hence, the sparsity of these vectors introduces many noise increasing the distances.

Figure 30 shows the evolution of the percentage of avoided comparisons and *mAP* applying the chosen parameters to the validation set. Observe that the performance decreases rapidly as more comparisons are avoided.

Until now, the validation set has been used to understand the behavior of the parameters. Table XV shows a comparison in

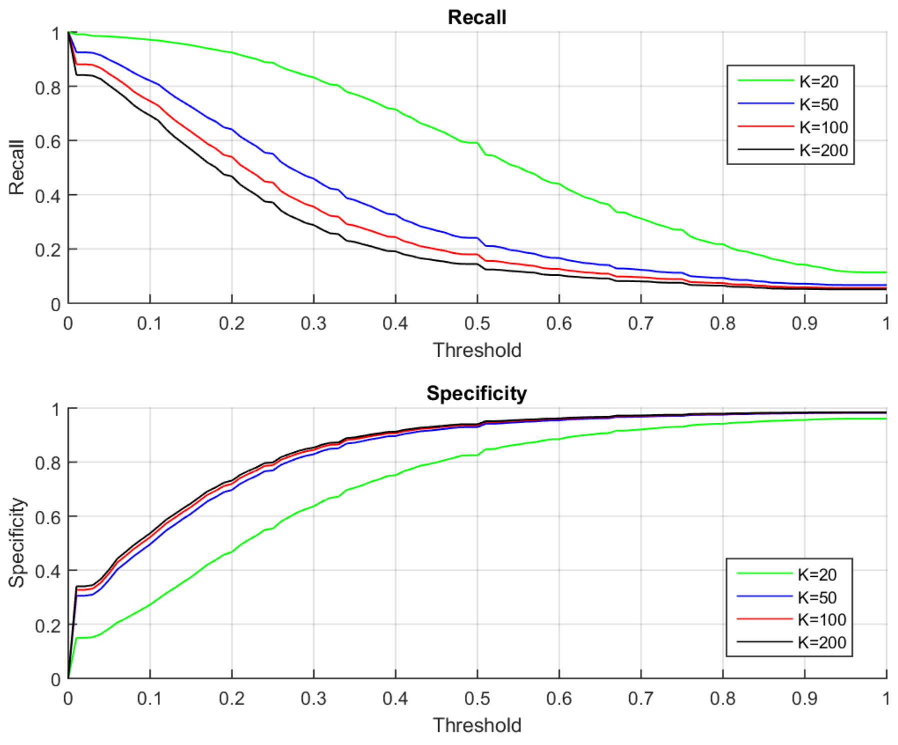


Fig. 29. Recall and specificity evolution depending on the threshold for some values of the graph labels.

TABLE XIV
SPECIFICITY FOR A FIXED RECALL AND *local context*

Configuration	**Specificity**	
#Clusters	Recall = 0.8	Recall = 0.7
20	**0.67**	**0.77**
50	0.52	0.63
100	0.45	0.57
200	0.40	0.54

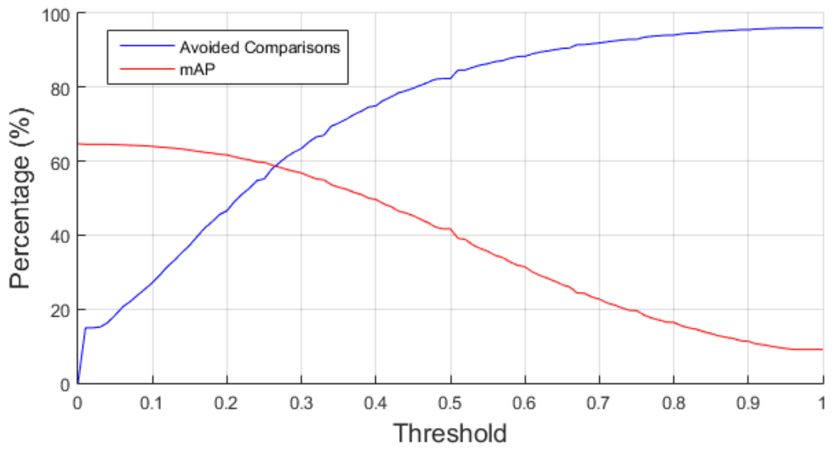


Fig. 30. Avoided comparisons and *mAP* evolution depending on the threshold used.

terms of *mAP*, *recall*, *specificity* and *time* using or not the indexation approach and two different thresholds. In comparison with the former experiment using the hierarchical representation, +Hierarchy (thresh=0.30) against +Indexation (thresh=0.20), we can observe that despite having the higher recall, the *mAP* is lower. It occurs because the indexation is not able to prune as many graphs as the hierarchy.

TABLE XV
COMPARISON WITH THE ORIGINAL MATCHING

	mAP (%)	**Recall (%)**	**Specificity (%)**	**Time per query[3] (s)**
Original	69.45	100	0	19.58
+Hierarchy (thresh=0.30)	68.27	90.91	69.98	12.46
+Hierarchy (thresh=0.25)	61.71	67.93	97.91	3.94
+Indexation (thresh=0.20)	66.13	92.54	46.13	16.34
+Indexation (thresh=0.30)	61.15	83.55	63.04	12.74

The interested reader can refer to [48], [49] where further experimentation is presented.

VI. Conclusions

This work has presented two contributions in the area of graph-based representations and matching. The first contribution is the construction of a hierarchical graph representation by means of contraction and embedding functions. The proposed contraction function uses graph clustering techniques to gather related nodes. Moreover, a modification of the contraction function has been proposed to stabilize the hierarchy in certain graphs. The developed framework is able to significantly reduce the graph size allowing a fast graph comparison. The information encoded in each abstract level enables a coarse-to-fine matching that prunes the amount of comparisons that must be done in the fine level. Moreover, given a level of the hierarchy, the second contribution is the enrichment of the node information with a Morgan Index based vector. This vector embeds information about the topology of their local context. Afterwards, a binarization of these vectors is proposed in order to create an indexation framework. Hence, the indexation approach allows to find the candidate graphs to match in a fast way. The contributions have been exhaustively validated using several databases, some of them corresponding to real applications of large-scale retrieval.

Compared to other related works, the proposed approach dynamically gathers the nodes without predefining the number of clusters, therefore, the number of levels in the hierarchy can be different from two graphs. Furthermore, the graph size is extremely reduced from one level to another.

From this work, we conclude that hierarchical graph representations are a powerful tool that can help in the matching process. This kind of representations give information about the relation of a group of nodes (those that are contracted) instead of the typical pair-wise relations that can be found in graph-based representations. Also, information between the relation of groups of nodes is provided through the edges that appear in each level. Moreover, the indexation step leads us to conclude that it is very useful to compute inexact subgraph matchings in large-scale scenarios as a filtering step for pruning the database, before using a more accurate matching method only in the retrieved subgraphs. Finally, in terms of the application, we have demonstrated that compact structural descriptors are useful signatures for handwriting recognition, despite the variability of handwriting.

The future work will be focused on the development of matching algorithms using the whole hierarchy at once. Moreover, a graph kernel or embedding will be proposed dealing with hierarchical representations.

Acknowledgment

I acknowledge my advisors Josep and Alicia for the patience, collaboration and enthusiasm during the development of this project, without their great guidance, this work would not be as it is today. Also thanks to Anjan, Marçal and Joan for the help provided in different aspects of this project. This work has been partially supported by the Spanish project TIN2015-70924-C2-2-R. The author also acknowledges the "Fundació Catalunya La Pedrera" for their grant "Beques Màsters d'Excellencia". Finally, special thanks to my friends and my family who have encouraged me during this project.

References

[1] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain, "Content-based multimedia information retrieval: State of the art and challenges," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 2, no. 1, pp. 1–19, 2006.

[2] M. Fischler and R. Elschlager, "The representation and matching of pictorial structures," *IEEE Transactions on Computers*, vol. C-22, no. 1, pp. 67–92, Jan 1973.

[3] K. Riesen and H. Bunke, *Graph Classification and Clustering Based on Vector Space Embedding*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2010.

[4] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 18, no. 03, pp. 265–298, 2004.

[5] A. Sanfeliu and K.-S. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-13, no. 3, pp. 353–362, May 1983.

[6] H. Bunke and G. Allermann, "Inexact graph matching for structural pattern recognition," *Pattern Recognition Letters*, vol. 1, no. 4, pp. 245–253, 1983.

[7] X. Gao, B. Xiao, D. Tao, and X. Li, "A survey of graph edit distance," *Pattern Analysis and applications*, vol. 13, no. 1, pp. 113–129, 2010.

[3] 1000 queries selected randomly against 13098 graphs

[8] K. Riesen and H. Bunke, "Approximate graph edit distance computation by means of bipartite graph matching," *Image and Vision Computing*, vol. 27, no. 7, pp. 950–959, 2009.

[9] F. Zhou and F. de la Torre, "Factorized graph matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, no. 99, pp. 1–1, 2015.

[10] H. Bunke and K. Riesen, "Towards the unification of structural and statistical pattern recognition," *Pattern Recognition Letters*, vol. 33, no. 7, pp. 811 – 825, 2012, special Issue on Awards from {ICPR} 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865511001309

[11] K. Riesen, M. Neuhaus, and H. Bunke, "Graph embedding in vector spaces by means of prototype selection," in *Graph-Based Representations in Pattern Recognition*. Springer, 2007, pp. 383–393.

[12] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl 1, pp. i47–i56, 2005.

[13] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *5th IEEE International Conference on Data Mining*, 2005, pp. 8–pp.

[14] Z. Harchaoui and F. Bach, "Image classification with segmentation graph kernels," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[15] D. Shasha, J. T. L. Wang, and R. Giugno, "Algorithmics and applications of tree and graph searching," in *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '02. New York, NY, USA: ACM, 2002, pp. 39–52. [Online]. Available: http://doi.acm.org/10.1145/543613.543620

[16] X. Yan, P. Yu, and J. Han, "Graph indexing: a frequent structure-based approach," in *Proceedings of the ACM SIGMOD international conference on Management of data*, 2004, pp. 335–346.

[17] B. Messmer and H. Bunke, "A decision tree approach to graph and subgraph isomorphism detection," *Pattern Recognition*, vol. 32, no. 12, pp. 1979 – 1998, 1999.

[18] E. Saund, "A graph lattice approach to maintaining and learning dense collections of subgraphs as image features," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 10, pp. 2323–2339, Oct. 2013.

[19] J. Cheng, Y. Ke, W. Ng, and A. Lu, "Fg-index: Towards verification-free query processing on graph databases," in *International Conference on Management of Data*, ser. SIGMOD '07. New York, NY, USA: ACM, 2007, pp. 857–872.

[20] S. E. Schaeffer, "Graph clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27–64, 2007.

[21] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002. [Online]. Available: http://www.pnas.org/content/99/12/7821.abstract

[22] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, pp. 35–41, 1977.

[23] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide-baseline stereo from maximally stable extremal regions," *Image and vision computing*, vol. 22, no. 10, pp. 761–767, 2004.

[24] D. W. Eggert, K. W. Bowyer, C. R. Dyer, H. I. Christensen, and D. B. Goldgof, "The scale space aspect graph," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 11, pp. 1114–1130, 1993.

[25] M. Ulrich, C. Wiedemann, and C. Steger, "Combining scale-space and similarity-based aspect graphs for fast 3d object recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 10, pp. 1902–1914, 2012.

[26] N. Ahuja and S. Todorovic, "From region based image representation to object discovery and recognition," in *Structural, Syntactic, and Statistical Pattern Recognition*. Springer, 2010, pp. 1–19.

[27] K. Broelemann, A. Dutta, X. Jiang, and J. Lladós, *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop, SSPR&SPR 2012, Hiroshima, Japan, November 7-9, 2012. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ch. Hierarchical Graph Representation for Symbol Spotting in Graphical Document Images, pp. 529–538. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34166-3_58

[28] S. F. Mousavi, M. Safayani, A. Mirzaei, and H. Bahonar, "Hierarchical graph embedding in vector space by graph pyramid," *Pattern Recognition*, vol. 61, pp. 245 – 254, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S003132031630200X

[29] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Calonder, michael and lepetit, vincent and strecha, christoph and fua, pascal," in *European Conference on Computer Vision*, ser. Lecture Notes on Computer Science. Springer, 2010, vol. 6314, pp. 778–792.

[30] H. L. Morgan, "The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service." *Journal of Chemical Documentation*, vol. 5, no. 2, pp. 107–113, 1965.

[31] N. Dahm, H. Bunke, T. Caelli, and Y. Gao, "A unified framework for strengthening topological node features and its application to subgraph isomorphism detection," in *Graph-Based Representations in Pattern Recognition*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 7877, pp. 11–20. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-38221-5_2

[32] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, ser. STOC '98. ACM, 1998, pp. 604–613.

[33] P. Erdös and A. Rényi, "On random graphs, i," *Publicationes Mathematicae (Debrecen)*, vol. 6, pp. 290–297, 1959.

[34] S. Nayar, S. Nene, and H. Murase, "Columbia object image library (coil 100)," *Department of Comp. Science, Columbia University, Tech. Rep. CUCS-006-96*, 1996.

[35] C. Harris and M. Stephens, "A combined corner and edge detector." in *Alvey vision conference*, vol. 15. Citeseer, 1988, p. 50.

[36] D. Fernández-Mota, J. Almazán, N. Cirera, A. Fornés, and J. Lladós, "Bh2m: The barcelona historical handwritten marriages database," *International Conference on Pattern Recognition*, 2014.

[37] S. Escalera, A. Fornés, O. Pujol, P. Radeva, G. Sánchez, and J. Lladós, "Blurred shape model for binary and grey-level symbol recognition," *Pattern Recognition Letters*, vol. 30, no. 15, pp. 1424 – 1433, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865509002074

[38] P. Riba, A. Fornés, and J. Lladós, "Handwritten word spotting by inexact matching of grapheme graphs," in *13th International Conference on Document Analysis and Recognition*, Aug 2015, pp. 781–785.

[39] M. Rusiñol and J. Lladós, "A performance evaluation protocol for symbol spotting systems in terms of recognition and location indices," *International Journal on Document Analysis and Recognition*, vol. 12, no. 2, pp. 83–96, 2009.

[40] J. Almazán, A. Gordo, A. Fornés, and E. Valveny, "Word spotting and recognition with embedded attributes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 12, pp. 2552–2566, Dec 2014.

[41] M. Rusiñol, D. Aldavert, R. Toledo, and J. Lladós, "Efficient segmentation-free keyword spotting in historical document collections," *Pattern Recognition*, vol. 48, no. 2, pp. 545–555, 2015.

[42] J. Lladós, M. Rusiñol, A. Fornés, D. Fernández, and A. Dutta, "On the influence of word representations for handwritten word spotting in historical documents," *Pattern Recognition and Artificial Intelligence, International Journal of*, vol. 26, no. 05, 2012.

[43] P. Wang, V. Eglin, C. Garcia, C. Largeron, J. Lladós, and A. Fornés, "A novel learning-free word spotting approach based on graph representation," in *11th IAPR International Workshop on Document Analysis Systems*, April 2014, pp. 207–211.

[44] A. Vinciarelli and S. Bengio, "Offline cursive word recognition using continuous density hidden markov models trained with pca or ica features," in *In proceedings of the 16th International Conference on Pattern Recognition*, vol. 3. IEEE, 2002, pp. 81–84.

[45] J. Almazán, A. Gordo, A. Fornés, and E. Valveny, "Efficient exemplar word spotting." in *In Proceedings of the British Machine Vision Conference*, vol. 1, no. 2, 2012, p. 3.

[46] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1201–1242, 2010.

[47] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt, "Efficient graphlet kernels for large graph comparison." in *12th International Conference on Artificial Intelligence and Statistics*, vol. 5, 2009, pp. 488–495.

[48] P. Riba, J. Lladós, A. Fornés, and A. Dutta, "Large-scale graph indexing using binary embeddings of node contexts," in *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 2015, pp. 208–217.

[49] P. Riba, J. Lladós, A. Fornés, and A. Dutta, "Large-scale graph indexing using binary embeddings of node contexts for information spotting in document image databases," *Pattern Recognition Letters*, pp. –, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865516301398